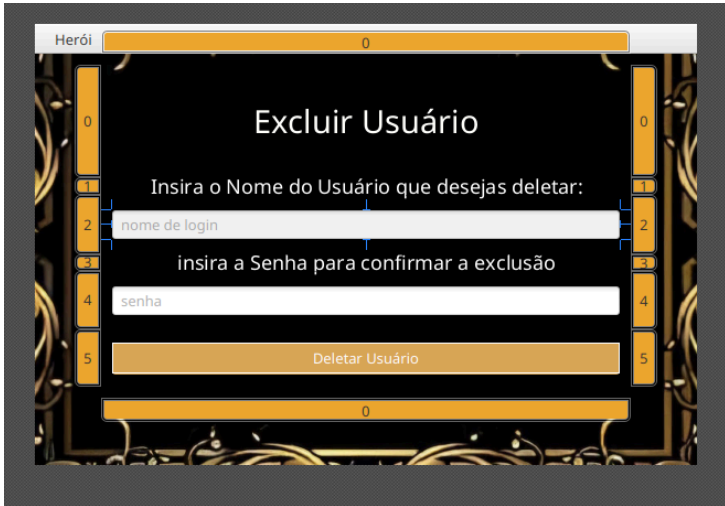
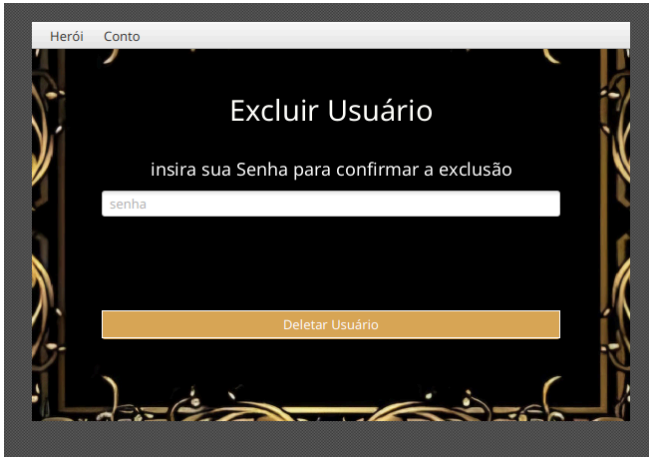
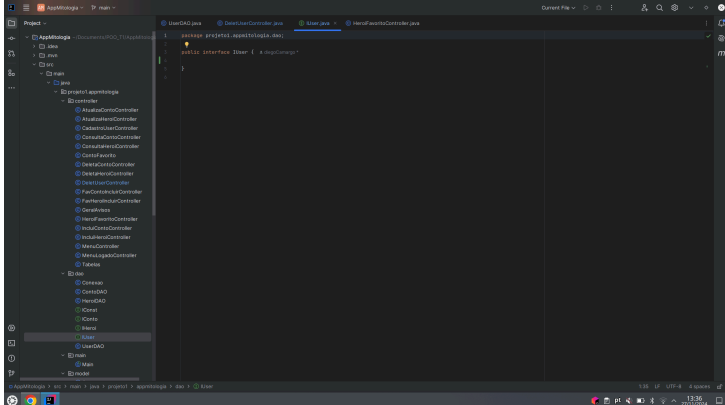


Diego

Nº Refatoração: 1	Classes: Session.java, DeletUserController.java, MenuController.java, UserDAO.java	Nome: refatoração encapsulamento session
Código antes de refatorar		Código após ser refatorado:
<pre>1 package projeto1.appmitologia.model; 2 3 public class Session { @diegoCamargo* 4 public static String cookie; 5 usages 5 } 6</pre>		<pre>1 package projeto1.appmitologia.model; 2 3 public class Session { @diegoCamargo* 4 public static String cookie; 7 usages 5 public static String getCookie() { no usages new* 6 return cookie; 7 } 8 9 public static void setCookie(String cookie) { no usages new* 10 Session.cookie = cookie; 11 } 12 }</pre>

Nº Refatoração: 2	Classes: DeletUserController.java,	Nome: não precisar inserir o nome de usuário para deletar um usuário
Código antes de refatorar		Código após ser refatorado:
<pre>23 public class DeletUserController { @diegoCamargo* 24 25 @FXML 26 private TextField userName; 27 @FXML 28 private PasswordField senha; 29 30 @FXML @diegoCamargo 31 void DeletUserOnAction(ActionEvent event) throws SQLException { 32 33 UserDAO userDAO = new UserDAO(); 34 String username = userName.getText(); 35 String pass = senha.getText(); 36 37 if (userDAO.authenticateUser(username, pass)) { 38 JOptionPane.showMessageDialog(parentComponent, null, message, "Usuário encontrado"); 39 40 UserDAO.remove(username); 41 Alert alertConfirmacao = new Alert(Alert.AlertType.INFORMATION, @ "Usuário deletado com sucesso!", ButtonType.OK); 42 alertConfirmacao.show(); 43 44 } else { 45 JOptionPane.showMessageDialog(parentComponent, null, message, "Invalid Credentials"); 46 } 47 } 48 }</pre>		<pre>public class DeletUserController { @diegoCamargo* @FXML private PasswordField senha; //Refatoração: não precisar inserir o nome de usuário para deletar um usuário @FXML @diegoCamargo* void DeletUserOnAction(ActionEvent event) throws SQLException { UserDAO userDAO = new UserDAO(); String pass = senha.getText(); if (userDAO.authenticateUser(Session.getCookie(), pass)) { UserDAO.remove(Session.getCookie()); Alert alertConfirmacao = new Alert(Alert.AlertType.INFORMATION, @ "Usuário deletado com sucesso!", ButtonType.OK); alertConfirmacao.show(); } else { JOptionPane.showMessageDialog(parentComponent, null, message, "Invalid Credentials"); } } }</pre>

Nº Refatoração: 3	Classes: deletUser.fxml,	Nome: Exclusão do campo de inserir o nome na interface
Código antes de refatorar		Código após ser refatorado:
		

Nº Refatoração: 4	Classes: ContoFavorito.java, ContoFavorito.java, HeroiFavoritoController.java, IUser.java,	Nome: Adição dos metodos a interface
Código antes de refatorar		Código após ser refatorado:
		<pre> 1 package projeto1.appnitologia.dao; 2 3 import projeto1.appnitologia.model.User; 4 import java.sql.SQLException; 5 6 public interface IUser { // implementation: &diegoCamargo 7 8 public void insert(User user) throws SQLException; //usage: implementation.new* 9 public boolean authenticateUser(String username, String password) throws SQLException; //usages: implementation.new* 10 public void favoriteHeroi(int idHeroi) throws SQLException; //usage: implementation.new* 11 public void favoriteConto(int idConto) throws SQLException; //usage: implementation.new* 12 public void remove(String id) throws SQLException; //usage: implementation.new* 13 public int getContoFav() throws SQLException; //usage: implementation.new* 14 public int getHeroiFav() throws SQLException; //usage: implementation.new* 15 } 16 17 </pre>

Ana Beatriz

Nº Refatoração: 1	Classes: deletaHeroi, deletaConto, consultaHeroi, consultaConto	Nome: Extrair métodos para classe Tabelas
Código antes de refatorar		Código após ser refatorado:
<p>Este código era repetido em todas as classes que o utilizavam</p> <pre> public TableCell<Heroi, String> wrap() { 2 usages Ana Beatriz return new TableCell<Heroi, String>() { 1 Ana Beatriz Text text = new Text(); 3 usages @Override 7 usages Ana Beatriz protected void updateItem(String item, boolean empty) { super.updateItem(item, empty); if (empty) { setGraphic(null); return; } text.setWrappingWidth(getTableColumn().getWidth() - 10); text.setText(item); setGraphic(text); } }; } </pre>		<pre> public class Tabelas { 4 usages public TableCell<Heroi, String> wrapHeroi() { 4 usages return new TableCell<Heroi, String>() { Text text = new Text(); 3 usages @Override 7 usages protected void updateItem(String item, boolean empty) { super.updateItem(item, empty); if (empty) { setGraphic(null); return; } text.setWrappingWidth(getTableColumn().getWidth() - 10); text.setText(item); setGraphic(text); } }; } public TableCell<Conto, String> wrapConto() { 4 usages return new TableCell<Conto, String>() { Text text = new Text(); 3 usages @Override 7 usages protected void updateItem(String item, boolean empty) { super.updateItem(item, empty); if (empty) { setGraphic(null); return; } text.setWrappingWidth(getTableColumn().getWidth() - 10); text.setText(item); setGraphic(text); } }; } } </pre>

2)

Nº Refatoração: 2	Classes: contoDAO	Nome: Extrair código para método
Código antes de refatorar		Código após ser refatorado:

<pre> public ArrayList<Conto> listaTodosNomeConto(String nomeConto) throws SQLException { open(); sql = "SELECT * FROM conto WHERE nomeConto = ?"; statement = connection.prepareStatement(sql); System.out.println(sql); result = statement.executeQuery(); ArrayList<Conto> contos = new ArrayList<>(); while (result.next()) { Conto conto = new Conto(); conto.setId(result.getInt(columnLabel: "contoId")); conto.setNome(result.getString(columnLabel: "nomeConto")); conto.setDescricao(result.getString(columnLabel: "descricaoConto")); conto.setLocalizacao(result.getString(columnLabel: "localizacaoConto")); Heroi heroi = new Heroi(); HeroiDAO heroiDAO = new HeroiDAO(); heroi.setId(result.getInt(columnLabel: "heroiId")); heroi = heroiDAO.buscaPorId(heroi.getId()); conto.setNameHeroi(heroi.getNome()); contos.add(conto); } close(); return contos; } public ArrayList<Conto> listaTodosNomesHeroi(String nomeHeroi) throws SQLException { open(); Heroi heroi = new Heroi(); HeroiDAO heroiDAO = new HeroiDAO(); heroi = heroiDAO.buscaPorNome(nomeHeroi); sql = "SELECT * FROM conto WHERE heroiId = ?"; statement = connection.prepareStatement(sql); statement.setInt(parameterIndex: 1, heroi.getId()); result = statement.executeQuery(); ArrayList<Conto> contos = new ArrayList<>(); while (result.next()) { Conto conto = new Conto(); conto.setId(result.getInt(columnLabel: "contoId")); conto.setNome(result.getString(columnLabel: "nomeConto")); conto.setDescricao(result.getString(columnLabel: "descricaoConto")); conto.setLocalizacao(result.getString(columnLabel: "localizacaoConto")); heroi.setId(result.getInt(columnLabel: "heroiId")); heroi = heroiDAO.buscaPorId(heroi.getId()); conto.setNameHeroi(heroi.getNome()); contos.add(conto); } close(); return contos; } </pre>	<pre> public ArrayList<Conto> listaTodosNomeConto(String nomeConto) throws SQLException { open(); sql = "SELECT * FROM conto WHERE nomeConto = ?"; return getArrayContos(); } public ArrayList<Conto> getArrayContos() throws SQLException { statement = connection.prepareStatement(sql); result = statement.executeQuery(); ArrayList<Conto> contos = new ArrayList<>(); while (result.next()) { Conto conto = new Conto(); conto.setId(result.getInt(columnLabel: "contoId")); conto.setNome(result.getString(columnLabel: "nomeConto")); conto.setDescricao(result.getString(columnLabel: "descricaoConto")); conto.setLocalizacao(result.getString(columnLabel: "localizacaoConto")); Heroi heroi = new Heroi(); HeroiDAO heroiDAO = new HeroiDAO(); heroi.setId(result.getInt(columnLabel: "heroiId")); heroi = heroiDAO.buscaPorId(heroi.getId()); conto.setNameHeroi(heroi.getNome()); contos.add(conto); } close(); return contos; } </pre>
--	--

3)

Nº Refatoração: 3	Classes: heroiDAO	Nome: Extrair código duplicado para método
Código antes de refatorar		Código após ser refatorado:
<pre> public Heroi buscaPorNome(String nomeHeroi) throws SQLException { open(); sql = "SELECT * FROM heroi WHERE nomeHeroi = ?"; statement = connection.prepareStatement(sql); statement.setString(parameterIndex: 1, nomeHeroi); result = statement.executeQuery(); if (result.next()) { Heroi heroi = new Heroi(); heroi.setId(result.getInt(columnLabel: "heroiId")); heroi.setNome(result.getString(columnLabel: "nomeHeroi")); heroi.setDescricao(result.getString(columnLabel: "descricaoHeroi")); heroi.setUrl(result.getString(columnLabel: "imagemHeroi")); close(); return heroi; } else { close(); return null; } } </pre>	<pre> public ArrayList<Heroi> getHerois() throws SQLException { statement = connection.prepareStatement(sql); result = statement.executeQuery(); ArrayList<Heroi> herois = new ArrayList<>(); while (result.next()) { Heroi heroi = new Heroi(); heroi.setId(result.getInt(columnLabel: "heroiId")); heroi.setNome(result.getString(columnLabel: "nomeHeroi")); heroi.setDescricao(result.getString(columnLabel: "descricaoHeroi")); heroi.setUrl(result.getString(columnLabel: "imagemHeroi")); heroi.setUrl(result.getString(columnLabel: "imagemHeroi")); try { heroi.setImagemFisica(new ImageView(new Image(heroi.getUrl()))); } catch (IllegalArgumentException e) { heroi.setImagemFisica(new ImageView(new Image("https://i.dbb.co/9np82Y/notFound.png"))); } heroi.getImagem().setFitHeight(70); heroi.getImagem().setPreserveRatio(true); herois.add(heroi); } close(); return herois; } </pre>	

4) Mudança nome variável para algo mais intuitivo

Nº Refatoração: 3	Classes: consultaHeroi, deletaHeroi,deletaCon to,consultaConto	Nome: Mudança de nome de variável, para legibilidade
Código antes de refatorar		Código após ser refatorado:
<pre>public void btnConsultarActionPerformed(ActionEvent actionEvent) throws SQLException { HeroiDAO heroiDAO = new HeroiDAO(); ObservableList<Heroi> objList = FXCollections.observableArrayList(); preencherColunas(); if (txtNome.getText().isEmpty()) { ArrayList<Heroi> heroi = new ArrayList<>(); heroi = heroiDAO.listaTodosNomeConto(txtNome.getText()); objList.addAll(heroi); tblHeroi.setListData(objList); } else { if (txtId.getText().isEmpty()) { try { Heroi heroi = new Heroi(); heroi = heroiDAO.buscarPorId(Integer.parseInt(txtId.getText())); objList.add(heroi); tblHeroi.setListData(objList); } catch (NumberFormatException e) { alert.alert(); } alert = new Alert(AlertType.ERROR_MODAL, "O 'Id' é obrigatório e precisa ser um número inteiro. Exemplo: 1234567890"); alert.setTitle("Atenção: um número entre 1 e 9"); alert.setHeaderText("Informação"); alert.show(); tblHeroi.setListData(objList); } else { tblHeroi.setListData(objList); } } }</pre>		<pre>public void btnConsultarOnAction(ActionEvent actionEvent) throws SQLException { ArrayList<Conto> contos = new ArrayList<>(); ContoDAO contoDAO = new ContoDAO(); ObservableList<Conto> listaParaTabela = FXCollections.observableArrayList(); preencherColunas(); if (txtNome.getText().isEmpty()) { contos = contoDAO.listaTodosNomeConto(txtNome.getText()); listaParaTabela.addAll(contos); tblConto.setItems(listaParaTabela); } }</pre>

Giovana

1)

<p>Nº Refatoração: 1</p>	<p>Classes: IncluiHeroiController, IncluiContoController, AtualizaContoController, AtualizaHeroiController</p>	<p>Nome: Novo sistema de validação de id</p>
<p>Código antes de refatorar</p>		<p>Código após ser refatorado:</p>
<pre>@FXML private void bAtualizaOnAction(ActionEvent event) { if (idConto.getText().isEmpty()) { Alert alert = new Alert(Alert.AlertType.WARNING, "Por favor, insira o ID do conto para atualizar.", ButtonType.OK); alert.setTitle("ID do Conto Necessário"); alert.setHeaderText("Atenção"); alert.show(); return; } int id; try { id = Integer.parseInt(idConto.getText()); } catch (NumberFormatException e) { Alert alert = new Alert(Alert.AlertType.ERROR, "ID do Conto deve ser um número.", ButtonType.OK); alert.setTitle("Erro no ID"); alert.setHeaderText("Atenção"); alert.show(); return; } }</pre>		<pre>/* 1ª Refatoração Autor: Giovana Previsão antes do preenchimento do ID Objetivo: Anular toda a validação em uma única função em uma classe geral*/ public int validaId(String idText, String tipo) { if (idText.isEmpty()) { criarAlerta(alert.AlertType.WARNING, "Por favor, insira o ID do " + tipo + " para atualizar.", tipo + " Necessário").show(); return -1; } try { return Integer.parseInt(idText); } catch (NumberFormatException e) { criarAlerta(alert.AlertType.ERROR, "ID do " + tipo + " deve ser um número.", "Erro no ID").show(); return -1; } } /* 2ª Refatoração Autor: Giovana Utiliza a validação de id Objetivo: Deixa o código mais claro e mais compacto, já que não é preciso repetir os métodos*/ @FXML private void bAtualizaContoOnAction(ActionEvent event) { int id = validarId(idConto.getText(), "Conto"); if (id == -1) return; ContoDAO contoDAO = new ContoDAO(); @FXML private void bAtualizaHeroiOnAction(ActionEvent event) { int idHeroi = validarId(idHeroiField.getText(), "Herói"); if (idHeroi == -1) return; HeroiDAO heroiDAO = new HeroiDAO();</pre>

2)

Nº Refatoração: 2	Classes: AtualizaContoController, AtualizaHeroiController	Nome: Separa as validações de campo do restante do código
Código antes de refatorar		Código após ser refatorado:
<pre> try { Conto contoExistente = contoDAO.buscarPorId(id); if (contoExistente == null) { Alert alert = new Alert(Alert.AlertType.WARNING, "Conto com o ID " + id + " não encontrado.", ButtonType.OK); alert.setTitle("Conto não encontrado"); alert.setHeaderText("Atenção"); alert.show(); return; } if (!descricao.getText().isEmpty()) { contoExistente.setDescricao(descricao.getText()); } if (!localizacao.getText().isEmpty()) { contoExistente.setLocalizacao(localizacao.getText()); } if (!nomeConto.getText().isEmpty()) { contoExistente.setNome(nomeConto.getText()); } if (!nomeHeroi.getText().isEmpty()) { contoExistente.setNomeHeroi(nomeHeroi.getText()); } } </pre>		<pre> /* 2ª Refatoração Autor: Giovana Separa as validações de campo do restante do código Objetivo: Deixa o código mais claro e mais compacto*/ private void atualizarDadosHeroi(Heroi heroi) { if (!nome.getText().isEmpty()) heroi.setNome(nome.getText()); if (!desc.getText().isEmpty()) heroi.setDescricao(desc.getText()); if (!img.getText().isEmpty()) heroi.setUrl(img.getText()); } 18 19 20 21 22 23 24 25 26 try { Conto contoExistente = contoDAO.buscarPorId(id); if (contoExistente == null) { criarAlerta(Alert.AlertType.WARNING, "Conto com ID " + id + " não encontrado.", "Conto não encontrado").show(); } return; } /* 2ª Refatoração Autor: Giovana Separa as validações de campo do restante do código Objetivo: Deixa o código mais claro e mais compacto*/ private void atualizarDadosConto(Conto conto) { if (!descricao.getText().isEmpty()) conto.setDescricao(descricao.getText()); if (!localizacao.getText().isEmpty()) conto.setLocalizacao(localizacao.getText()); if (!nomeConto.getText().isEmpty()) conto.setNome(nomeConto.getText()); if (!nomeHeroi.getText().isEmpty()) conto.setNomeHeroi(nomeHeroi.getText()); } </pre>

3)

Nº Refatoração: 3	Classes: AtualizaHeroiController, AtualizaContoController, IncluiHeroiController, IncluiContoController	Nome: Criação de classe GeralAvisos para reunir a criação e exibição dos alerts
-------------------	---	---

Código antes de refatorar	Código após ser refatorado:
<pre> 69 - Alert confirmAlert = new Alert(Alert.AlertType.CONFIRMATION, "Deseja atualizar o Conto?", ButtonType.CANCEL, ButtonType.OK); 70 - confirmAlert.setTitle("Confirmação de Atualização"); 71 - confirmAlert.setHeaderText("Confirme a atualização"); 72 73 74 confirmAlert.showAndWait().ifPresent(response -> { 75 if (response == ButtonType.OK) { 76 try { 77 contoDAO.atualiza(contoExistente); 78 Alert successAlert = new Alert(Alert.AlertType.INFORMATION, "Conto atualizado com sucesso!", ButtonType.OK); 79 successAlert.setTitle("Sucesso"); 80 successAlert.setHeaderText("Informação"); 81 successAlert.show(); 82 } catch (SQLException e) { 83 mostrarErroAlert("Erro ao atualizar conto!"); 84 e.printStackTrace(); 85 } 86 } else { 87 Alert cancelAlert = new Alert(Alert.AlertType.INFORMATION, "Atualização do conto cancelada.", ButtonType.OK); 88 cancelAlert.setTitle("Cancelado"); 89 cancelAlert.setHeaderText("Informação"); 90 cancelAlert.show(); 91 } 92 }) </pre>	<pre> try { Conto contoExistente = contoDAO.buscarPorId(id); if (contoExistente == null) { criarAlerta(Alert.AlertType.WARNING, "Conto com ID " + id + " não encontrado.", "Conto não encontrado").show(); return; } atualizarDados(contoExistente); Alert confirmAlert = criarAlerta(Alert.AlertType.CONFIRMATION, "Deseja atualizar o Conto?", "Confirmação de Atualização"); confirmAlert.showAndWait().ifPresent(response -> { if (response == ButtonType.OK) { try { contoDAO.atualiza(contoExistente); criarAlerta(Alert.AlertType.INFORMATION, "Conto atualizado com sucesso!", "Sucesso").show(); } catch (SQLException e) { mostrarErroAlert("Erro ao atualizar conto!"); e.printStackTrace(); } } else { criarAlerta(Alert.AlertType.INFORMATION, "Atualização do conto cancelada.", "Cancelado").show(); } }); } catch (SQLException e) { mostrarErroAlert("Erro ao buscar conto!"); e.printStackTrace(); } /* 3ª Refatoração Autor: Giovana Reune a lógica de alertas em uma única classe. Utilizado em Atualiza* e Inclui* Objetivo: Utilizando essa classe, não é necessário repetir essas linhas em todas public void mostrarErroAlert(String mensagem) { criarAlerta(Alert.AlertType.ERROR, mensagem, "Erro").show(); } /* 3ª Refatoração Autor: Giovana Reune a lógica de alertas em uma única classe. Utilizado em Atualiza* e Inclui* Objetivo: Utilizando essa classe, não é necessário repetir essas linhas em todas public Alert criarAlerta(Alert.AlertType tipo, String mensagem, String titulo) { Alert alert = new Alert(tipo, mensagem, ButtonType.OK); alert.setTitle(titulo); alert.setHeaderText("Informação"); return alert; } </pre>

4)

Nº Refatoração: 4	Classes: IncluiHeroiController, IncluiContoController	Nome: Separa o preenchimento de informações
Código antes de refatorar		Código após ser refatorado:
<pre> @FXML private void bCadastroOnAction(ActionEvent event) { Heroi heroi = new Heroi(); heroi.setNome(nome.getText()); heroi.setDescricao(desc.getText()); heroi.setUrl(img.getText()); Alert alert = new Alert(Alert.AlertType.CONFIRMATION, "Deseja inserir o Herói?", ButtonType.CANCEL, ButtonType.OK); alert.setTitle("Herói pode ser cadastrado!"); alert.setHeaderText("Informação"); alert.showAndWait().ifPresent(response -> { public class IncluiContoController { @FXML private Button bCadastro; @FXML private TextField nome, localizacao, nomeHeroi; @FXML private TextArea descricao; @FXML private void bCadastroOnAction(ActionEvent event) { Conto conto = new Conto(); Alert alert = new Alert(Alert.AlertType.CONFIRMATION, "Deseja inserir o Conto?", ButtonType.CANCEL, ButtonType.OK); alert.setTitle("Conto pode ser cadastrado!"); alert.setHeaderText("Informação"); alert.showAndWait().ifPresent(response -> { </pre>		<pre> /* 4ª Refatoração Autor: Giovana Separa os preenchimentos de campo do restante do código Objetivo: Deixa o código mais claro e mais compacto*/ private Heroi criarHeroi() { Heroi heroi = new Heroi(); heroi.setNome(nome.getText()); heroi.setDescricao(desc.getText()); heroi.setUrl(img.getText()); return heroi; } /* 4ª Refatoração Autor: Giovana Separa os preenchimentos de campo do restante do código Objetivo: Deixa o código mais claro e mais compacto*/ private Conto criarConto() { Conto conto = new Conto(); conto.setNome(nome.getText()); conto.setDescricao(descricao.getText()); conto.setLocalizacao(localizacao.getText()); conto.setNomeHeroi(nomeHeroi.getText()); return conto; } </pre>

