LINGUAGEM DE PROGRAMAÇÃO ORIENTADA A OBJETOS

ETEC DE HORTOLÂNDIA

CURSO TÉCNICO EM INFORMÁTICA INTEGRADO AO ENSINO MÉDIO

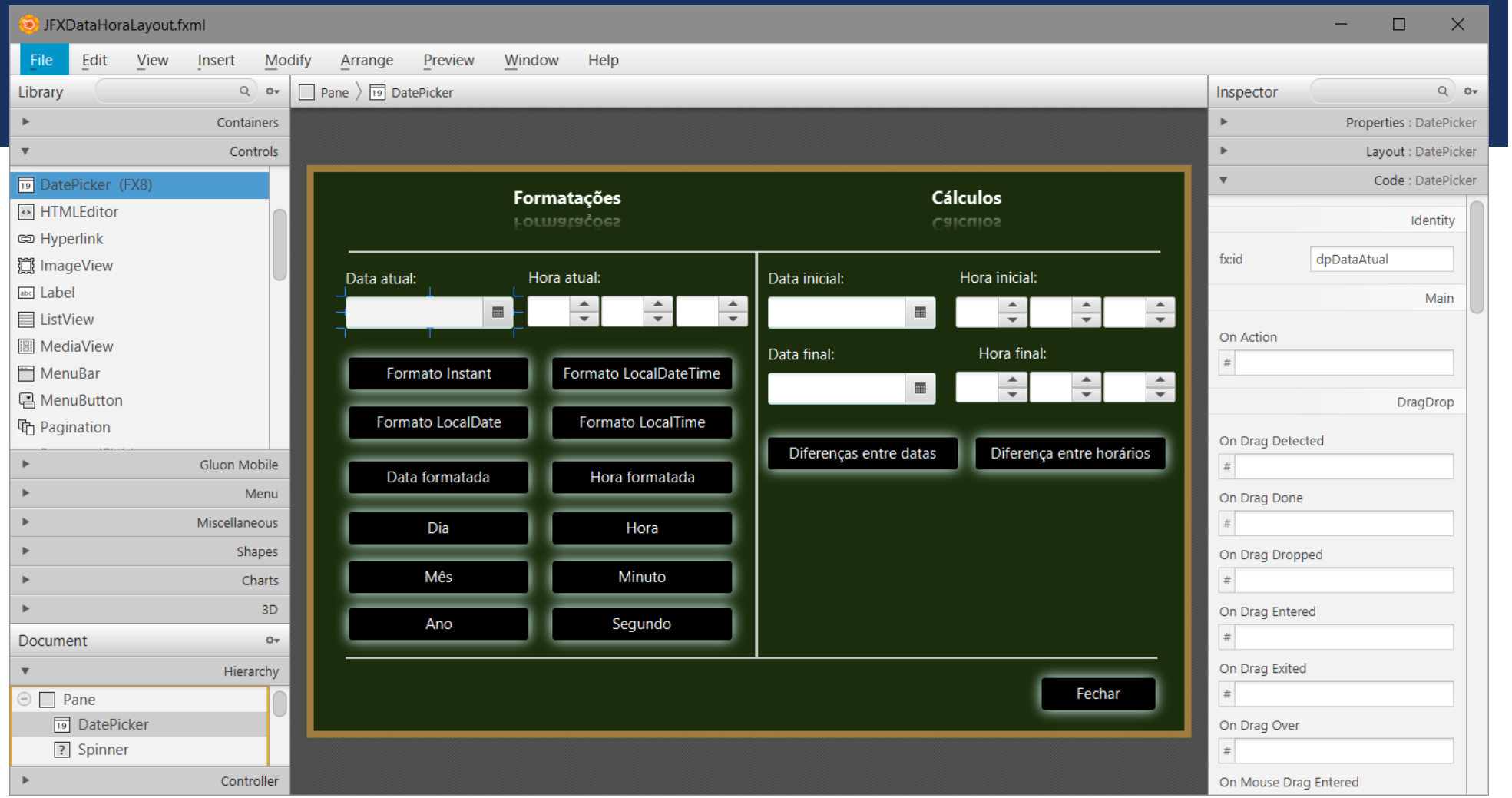
PROF. RALFE DELLA CROCE FILHO

CONTEÚDO

- Componentes JavaFX
  - DatePicker
  - Spinner
- Classes
  - Instant
  - LocalDateTime
  - LocalDate
  - LocalTime
  - DateTimeFormatter
  - Period
  - Duration

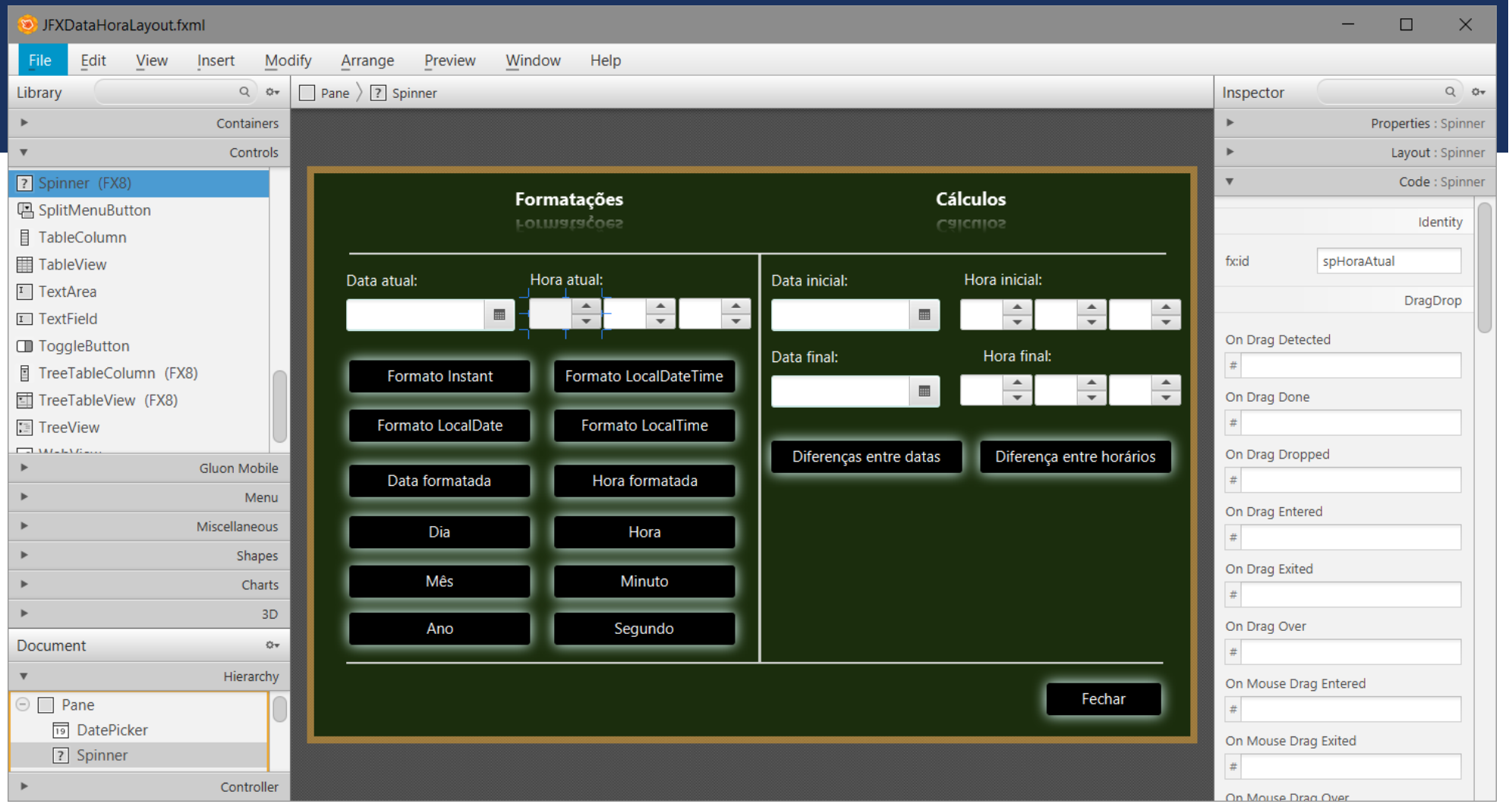
LAYOUT

- DatePicker

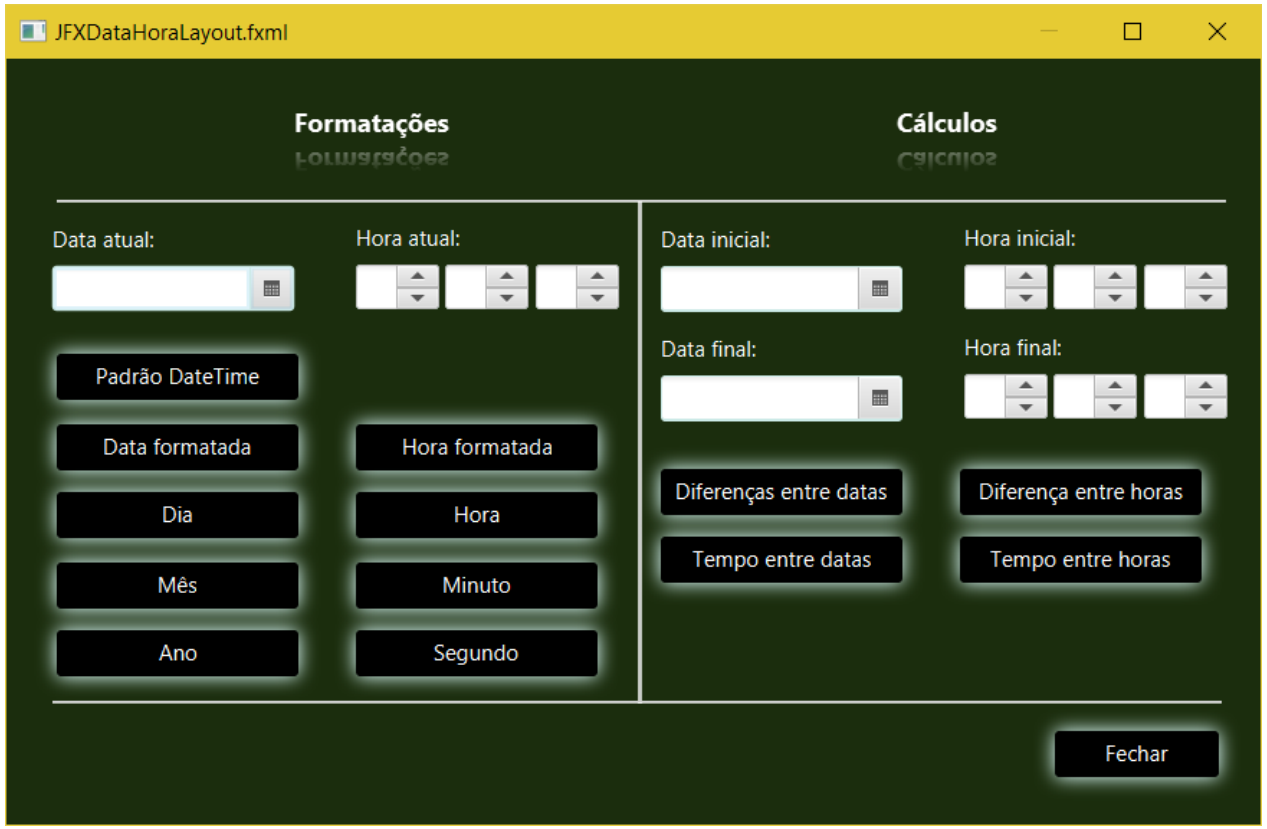


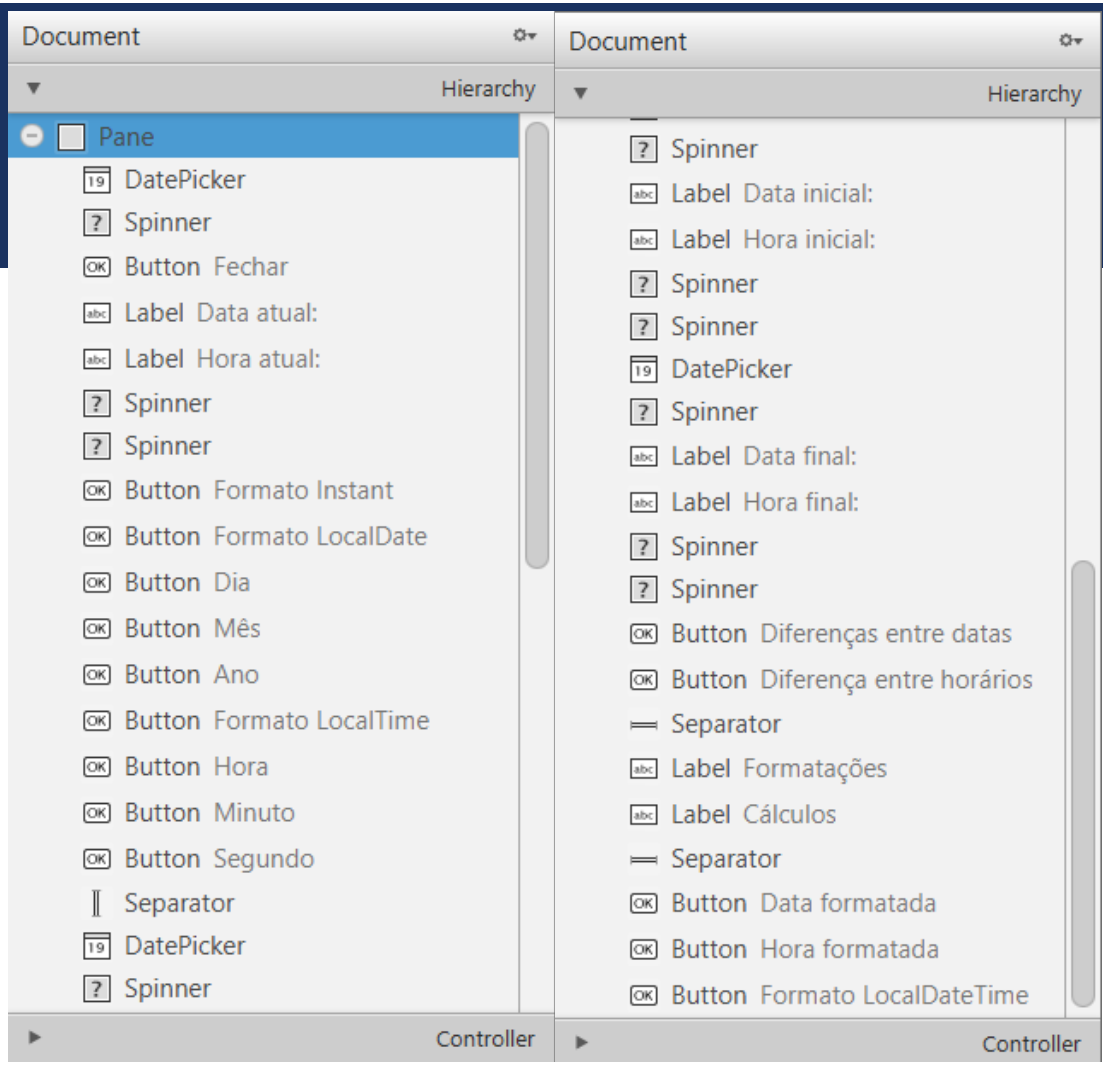
LAYOUT

- Spinner



LAYOUT


JFXDataHoraLayout.fxml
Formatações
Cálculos
Data atual:
Hora atual:
Data inicial:
Hora inicial:
Data final:
Hora final:
Padrão DateTime
Data formatada
Dia
Mês
Ano
Hora formatada
Hora
Minuto
Segundo
Diferenças entre datas
Tempo entre datas
Diferença entre horas
Tempo entre horas
Fechar


Document
Hierarchy
Pane
DatePicker
Spinner
Button Fechar
Label Data atual:
Label Hora atual:
Spinner
Spinner
Button Formato Instant
Button Formato LocalDate
Button Dia
Button Mês
Button Ano
Button Formato LocalTime
Button Hora
Button Minuto
Button Segundo
Separator
DatePicker
Spinner
Controller
Document
Hierarchy
Spinner
Label Data inicial:
Label Hora inicial:
Spinner
Spinner
DatePicker
Spinner
Label Data final:
Label Hora final:
Spinner
Spinner
Button Diferenças entre datas
Button Diferença entre horários
Separator
Label Formatações
Label Cálculos
Separator
Button Data formatada
Button Hora formatada
Button Formato LocalDateTime
Controller

CONTROLE

JFXDataHoraControle.java

```
1  package br.resistencia.controller;
2
3  import java.net.URL;
4  import java.time.Duration;
5  import java.time.Instant;
6  import java.time.LocalDate;
7  import java.time.LocalDateTime;
8  import java.time.LocalTime;
9  import java.time.Period;
10 import java.time.format.DateTimeFormatter;
11 import java.util.ResourceBundle;
12
13 import javafx.fxml.FXML;
14 import javafx.fxml.Initializable;
15 import javafx.scene.control.Alert;
16 import javafx.scene.control.Alert.AlertType;
17 import javafx.scene.control.Button;
18 import javafx.scene.control.DatePicker;
19 import javafx.scene.control.Spinner;
20 import javafx.scene.control.SpinnerValueFactory;
21 import javafx.stage.Stage;
```

CONTROLE

JFXDataHoraControle.java

```java
public class JFXDataHoraControle implements Initializable{

    // Atributo
    private Stage palcoDataHora;
    // Controles FXML
    @FXML private DatePicker dpDataAtual;
    @FXML private Spinner<Integer> spHoraAtual;
    @FXML private Spinner<Integer> spMinutoAtual;
    @FXML private Spinner<Integer> spSegundoAtual;
    @FXML private DatePicker dpDataInicial;
    @FXML private Spinner<Integer> spHoraInicial;
    @FXML private Spinner<Integer> spMinutoInicial;
    @FXML private Spinner<Integer> spSegundoInicial;
    @FXML private DatePicker dpDataFinal;
    @FXML private Spinner<Integer> spHoraFinal;
    @FXML private Spinner<Integer> spMinutoFinal;
    @FXML private Spinner<Integer> spSegundoFinal;
    @FXML private Button bInstant;
    @FXML private Button bLocalDateTime;
    @FXML private Button bLocalDate;
    @FXML private Button bLocalTime;
    @FXML private Button bDataFormatada;
    @FXML private Button bDia;
    @FXML private Button bMes;
    @FXML private Button bAno;
    @FXML private Button bHoraFormatada;
    @FXML private Button bHora;
    @FXML private Button bMinuto;
    @FXML private Button bSegundo;
    @FXML private Button bDiferencaDatas;
    @FXML Button bDiferencaHorarios;
    @FXML private Button bFechar;
```

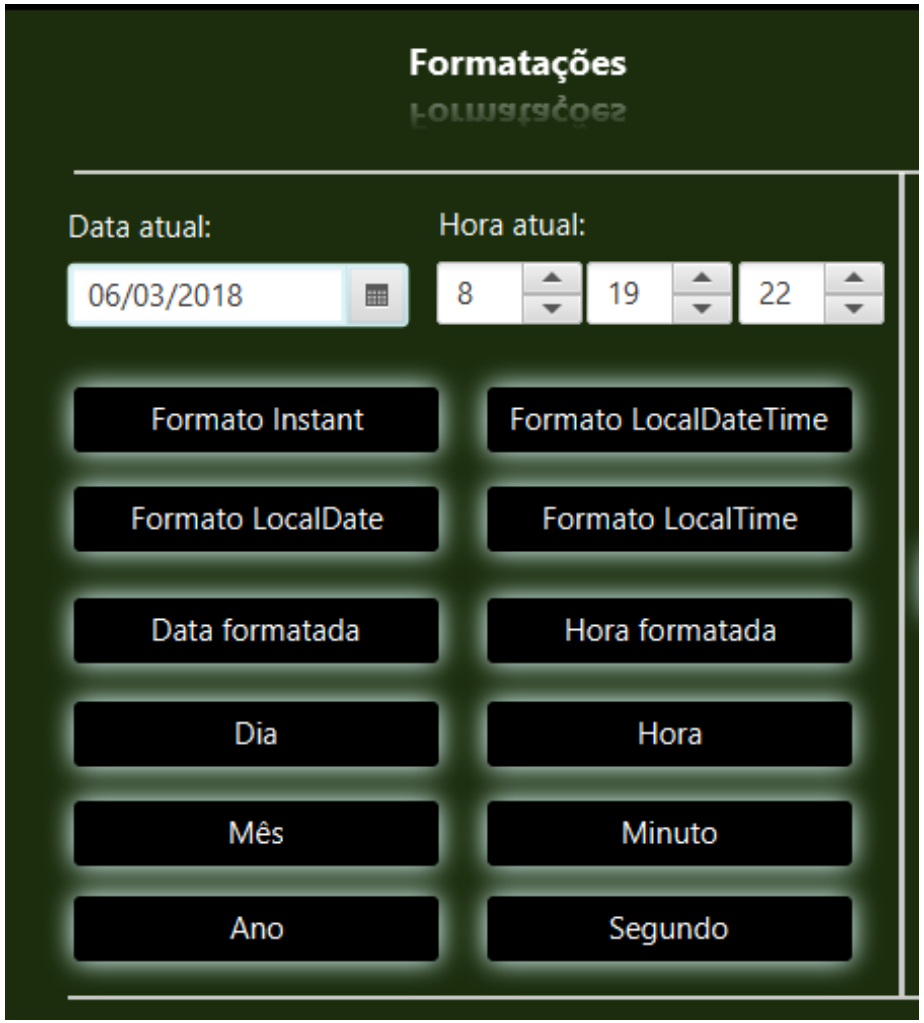
JFXDataHoraControle.java

```java
    // Metodos de acesso ao atributo
    public Stage getPalcoDataHora() {
        return palcoDataHora;
    }

    public void setPalcoDataHora(Stage palcoDataHora) {
        this.palcoDataHora = palcoDataHora;
    }
```

CONTROLE

JFXDataHoraControle.java

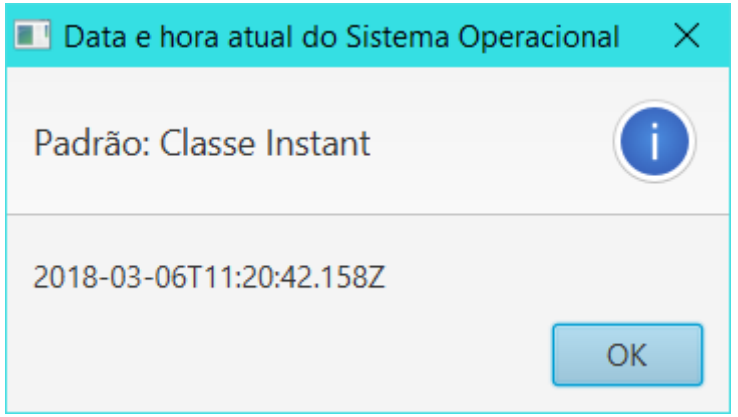
```java
    @Override
    public void initialize(URL location, ResourceBundle resources) {

        // Inicializa o controle DatePicker dpDataAtual com a data do Sistema Operacional
        dpDataAtual.setValue(LocalDate.now());

        // Inicializa os controles Spinners spHoraAtual, spMinutoAtual e spSegundoAtual, respectivamente,
        // com a hora, minuto e segundo do Sistema Operacional
        // e limitando os valores que poderão ser selecionados de acordo com seus imites
        SpinnerValueFactory<Integer> horaAtual = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 23, LocalTime.now().getHour());
        spHoraAtual.setValueFactory(horaAtual);
        SpinnerValueFactory<Integer> minutoAtual = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 59, LocalTime.now().getMinute());
        spMinutoAtual.setValueFactory(minutoAtual);
        SpinnerValueFactory<Integer> segundoAtual = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 59, LocalTime.now().getSecond());
        spSegundoAtual.setValueFactory(segundoAtual);

        // Inicializa os controles Spinners spHoraInicial, spMinutoInicial e spSegundoInicial
        // limitando os valores que poderão ser selecionados de acordo com seus imites
        SpinnerValueFactory<Integer> horaInicial = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 23, 0);
        spHoraInicial.setValueFactory(horaInicial);
        SpinnerValueFactory<Integer> minutoInicial = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 59, 0);
        spMinutoInicial.setValueFactory(minutoInicial);
        SpinnerValueFactory<Integer> segundoInicial = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 59, 0);
        spSegundoInicial.setValueFactory(segundoInicial);

        // Inicializa os controles Spinners spHoraFinal, spMinutoFinal e spSegundoFinal
        // limitando os valores que poderão ser selecionados de acordo com seus imites
        SpinnerValueFactory<Integer> horaFinal = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 23, 0);
        spHoraFinal.setValueFactory(horaFinal);
        SpinnerValueFactory<Integer> minutoFinal = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 59, 0);
        spMinutoFinal.setValueFactory(minutoFinal);
        SpinnerValueFactory<Integer> segundoFinal = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 59, 0);
        spSegundoFinal.setValueFactory(segundoFinal);
    }
```


Formatações
Data atual:
06/03/2018
Hora atual:
8
19
22
Formato Instant
Formato LocalDateTime
Formato LocalDate
Formato LocalTime
Data formatada
Hora formatada
Dia
Hora
Mês
Minuto
Ano
Segundo

CLASSE INSTANT

JFXDataHoraControle.java

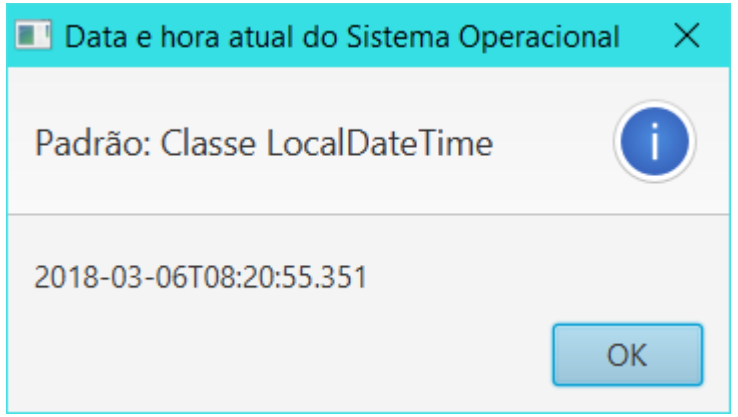
```
104     @FXML public void padraoInstant() {
105
106         // Hora do Sistema Operacional no formato Instant
107         Instant DataHoraAtualInstant = Instant.now();
108
109         Alert alert = new Alert(AlertType.INFORMATION);
110         alert.setTitle("Data e hora atual do Sistema Operacional");
111         alert.setHeaderText("Padrão: Classe Instant");
112         alert.setContentText( DataHoraAtualInstant.toString() );
113         alert.showAndWait();
114     }
```



CLASSE LOCALDATETIME

JFXDataHoraControle.java

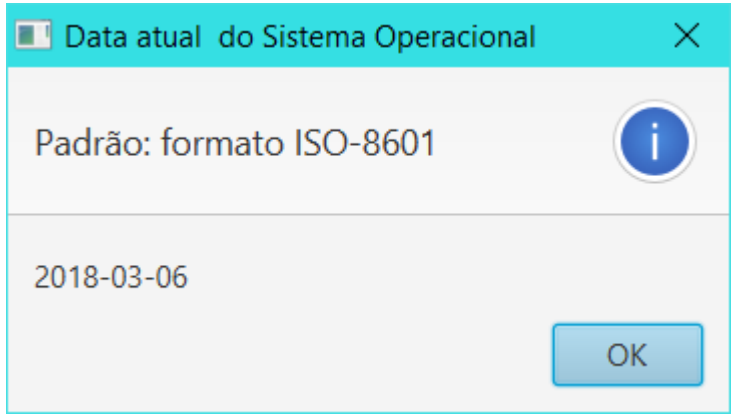
```
117     @FXML public void padraoLocalDateTime() {
118
119         // Hora do Sistema Operacional no formato LocalDateTime
120         LocalDateTime DataHoraAtualLocalDateTime = LocalDateTime.now();
121
122         Alert alert = new Alert(AlertType.INFORMATION);
123         alert.setTitle("Data e hora atual do Sistema Operacional");
124         alert.setHeaderText("Padrão: Classe LocalDateTime");
125         alert.setContentText( DataHoraAtualLocalDateTime.toString() );
126         alert.showAndWait();
127     }
```



CLASSE LOCALDATE

JFXDataHoraControle.java

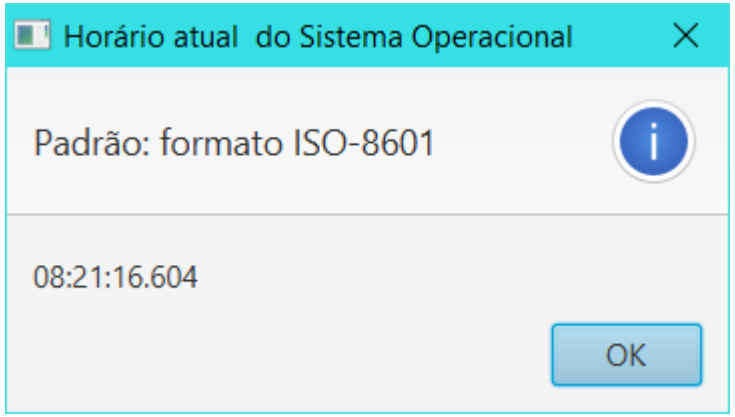
```
130     @FXML public void padraoLocalDate() {
131
132         // Hora do Sistema Operacional no formato LocalDate
133         LocalDate DataHoraAtualLocalDate = LocalDate.now();
134
135         Alert alert = new Alert(AlertType.INFORMATION);
136         alert.setTitle("Data atual  do Sistema Operacional");
137         alert.setHeaderText("Padrão: formato ISO-8601");
138         alert.setContentText( DataHoraAtualLocalDate.toString() );
139         alert.showAndWait();
140     }
```



CLASSE LOCALTIME

JFXDataHoraControle.java

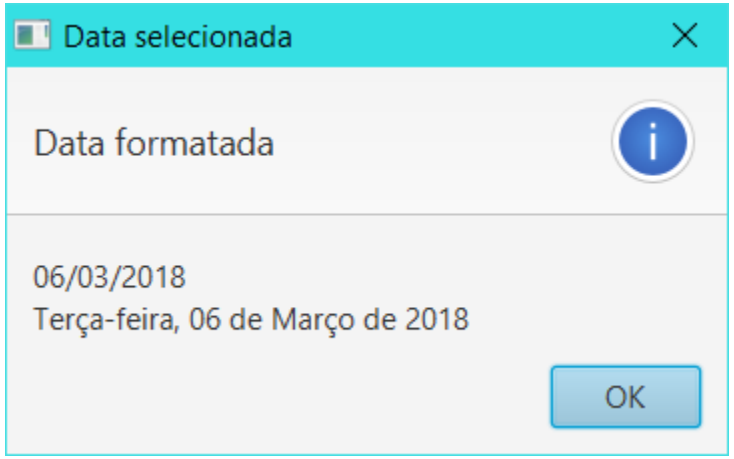
```
143     @FXML public void padraoLocalTime() {
144
145         // Hora do Sistema Operacional no formato LocalTime
146         LocalTime DataHoraAtualLocalTime = LocalTime.now();
147
148         Alert alert = new Alert(AlertType.INFORMATION);
149         alert.setTitle("Horário atual  do Sistema Operacional");
150         alert.setHeaderText("Padrão: formato ISO-8601");
151         alert.setContentText( DataHoraAtualLocalTime.toString() );
152         alert.showAndWait();
153     }
```



CLASSE DATETIMEFORMATTER

JFXDataHoraControle.java

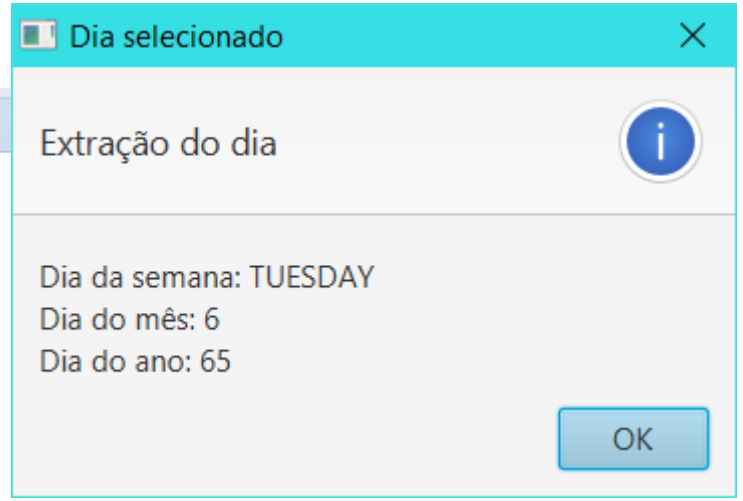
```
156     @FXML public void dataFormatada() {
157
158         // Formatações para a data lida a partir do controle DatePicker dpDataAtual
159         DateTimeFormatter formatador = DateTimeFormatter.ofPattern("dd/MM/yyyy");
160         DateTimeFormatter formatadorExtenso = DateTimeFormatter.ofPattern("EEEE, dd 'de' MMMM 'de' yyyy");
161
162         Alert alert = new Alert(AlertType.INFORMATION);
163         alert.setTitle("Data selecionada");
164         alert.setHeaderText("Data formatada");
165         alert.setContentText( dpDataAtual.getValue().format(formatador) + "\n" + dpDataAtual.getValue().format(formatadorExtenso) );
166         alert.showAndWait();
167     }
```



OBTER O DIA DE UMA DATA (LOCALDATE)

JFXDataHoraControle.java

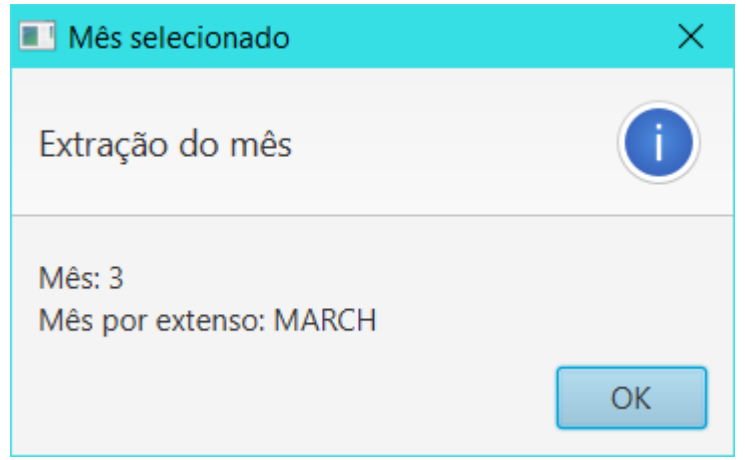
```
170     @FXML public void dia() {
171
172         // Obtém a data lida a partir do controle DatePicker dpDataAtual
173         LocalDate diaAtual = LocalDate.parse(dpDataAtual.getValue().toString());
174
175         // Extração do dia em diferentes formatos
176         String diaDaSemana = String.valueOf( diaAtual.getDayOfWeek() );
177         String diaDoMes = String.valueOf( diaAtual.getDayOfMonth() );
178         String diaDoAno = String.valueOf( diaAtual.getDayOfYear() );
179
180         Alert alert = new Alert(AlertType.INFORMATION);
181         alert.setTitle("Dia selecionado");
182         alert.setHeaderText("Extração do dia");
183         alert.setContentText("Dia da semana: " + diaDaSemana + "\nDia do mês: " + diaDoMes + "\nDia do ano: " + diaDoAno);
184         alert.showAndWait();
185     }
```



OBTER O MÊS DE UMA DATA (LOCALDATE)

JFXDataHoraControle.java

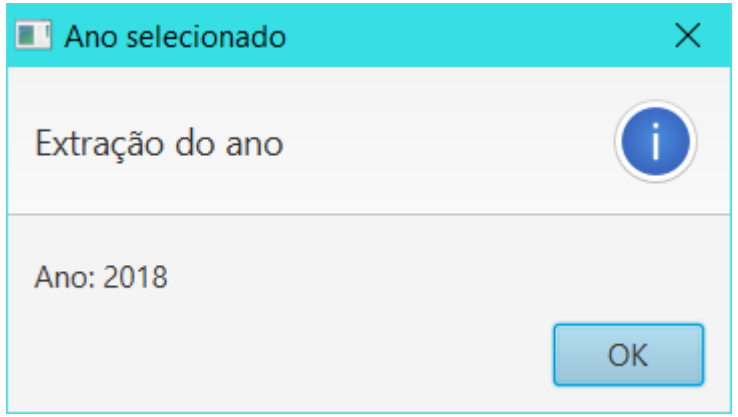
```
188    @FXML public void mes() {
189
190        // Obtém a data lida a partir do controle DatePicker dpDataAtual
191        LocalDate mesAtual = LocalDate.parse(dpDataAtual.getValue().toString());
192
193        // Extração do mês em diferentes formatos
194        String mesExtenso = String.valueOf( mesAtual.getMonth() );
195        String mes = String.valueOf( mesAtual.getMonthValue() );
196
197        Alert alert = new Alert(AlertType.INFORMATION);
198        alert.setTitle("Mês selecionado");
199        alert.setHeaderText("Extração do mês");
200        alert.setContentText( "Mês: " + mes + "\nMês por extenso: " + mesExtenso );
201        alert.showAndWait();
202    }
```



OBTER O ANO DE UMA DATA (LOCALDATE)

JFXDataHoraControle.java

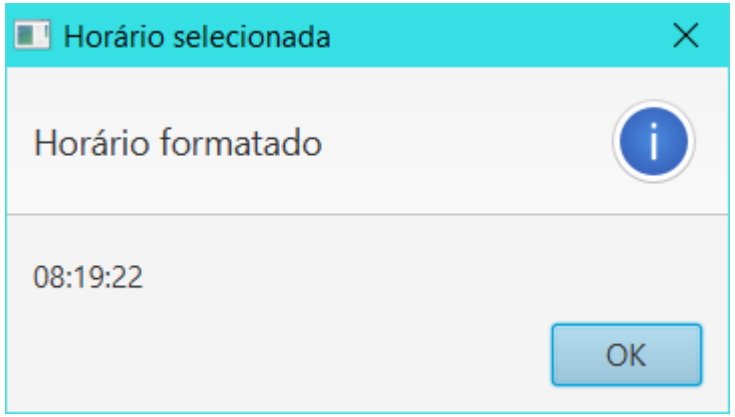
```
205     @FXML public void ano() {
206
207         // Obtém a data lida a partir do controle DatePicker dpDataAtual
208         LocalDate anoAtual = LocalDate.parse(dpDataAtual.getValue().toString());
209
210         // Extração do ano em diferentes formatos
211         String ano = String.valueOf( anoAtual.getYear() );
212
213         Alert alert = new Alert(AlertType.INFORMATION);
214         alert.setTitle("Ano selecionado");
215         alert.setHeaderText("Extração do ano");
216         alert.setContentText( "Ano: " + ano );
217         alert.showAndWait();
218     }
```



HORÁRIO FORMATADO (LOCALTIME)

JFXDataHoraControle.java

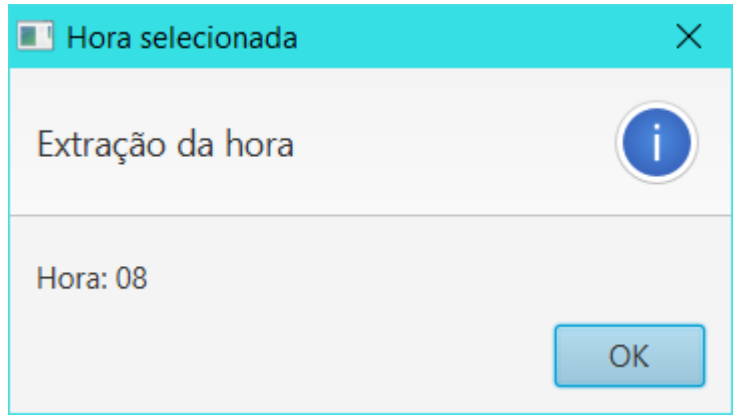
```
221    @FXML public void horaFormatada() {
222
223        // Obtém a hora, minuto e segundo lidos a partir dos controles Spinner spHoraAtual, spMinutoAtual e spSegundoAtual
224        // formatados com dois dígitos
225        String horaAtual = String.format("%02d", spHoraAtual.getValue());
226        String minutoAtual = String.format("%02d", spMinutoAtual.getValue());
227        String segundoAtual = String.format("%02d", spSegundoAtual.getValue());
228        // Monta uma String com o horário completo
229        String horarioAtualString = horaAtual + ":" + minutoAtual + ":" + segundoAtual;
230
231        // Converte o horário de String para LocalTime no formato HH:mm:ss
232        DateTimeFormatter formatador = DateTimeFormatter.ofPattern("HH:mm:ss");
233        LocalTime horarioAtualFormatado = LocalTime.parse(horarioAtualString, formatador);
234
235        Alert alert = new Alert(AlertType.INFORMATION);
236        alert.setTitle("Horário selecionada");
237        alert.setHeaderText("Horário formatado");
238        alert.setContentText( horarioAtualFormatado.toString() );
239        alert.showAndWait();
240    }
```



OBTER A HORA FORMATADA (LOCALTIME)

JFXDataHoraControle.java

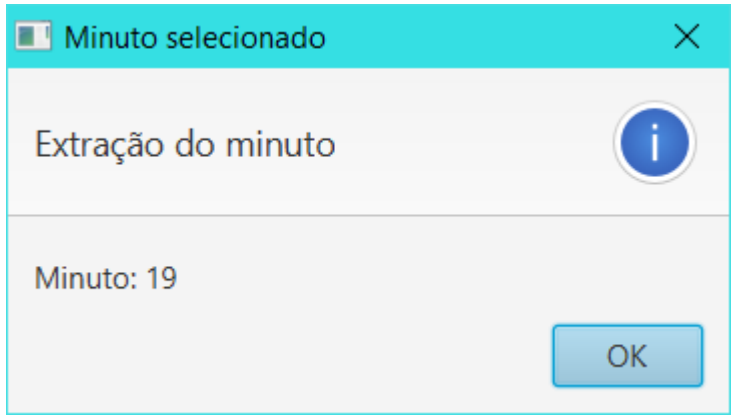
```
243    @FXML public void hora() {
244
245        // Obtém a hora, minuto e segundo lidos a partir dos controles Spinner spHoraAtual, spMinutoAtual e spSegundoAtual
246        // formatados com dois dígitos
247        String horaAtual = String.format("%02d", spHoraAtual.getValue());
248        String minutoAtual = String.format("%02d", spMinutoAtual.getValue());
249        String segundoAtual = String.format("%02d", spSegundoAtual.getValue());
250        // Monta uma String com o horário completo
251        String horarioAtualString = horaAtual + ":" + minutoAtual + ":" + segundoAtual;
252
253        // Converte o horário de String para LocalTime no formato HH:mm:ss
254        DateTimeFormatter formatador = DateTimeFormatter.ofPattern("HH:mm:ss");
255        LocalTime horarioAtualFormatado = LocalTime.parse(horarioAtualString, formatador);
256
257        Alert alert = new Alert(AlertType.INFORMATION);
258        alert.setTitle("Hora selecionada");
259        alert.setHeaderText("Extração da hora");
260        // Apresenta somente as horas formatada com dois dígitos
261        alert.setContentText( "Hora: " + String.valueOf( String.format("%02d", horarioAtualFormatado.getHour() )));
262        alert.showAndWait();
263    }
```



OBTER O MINUTO FORMATADO (LOCALTIME)

JFXDataHoraControle.java

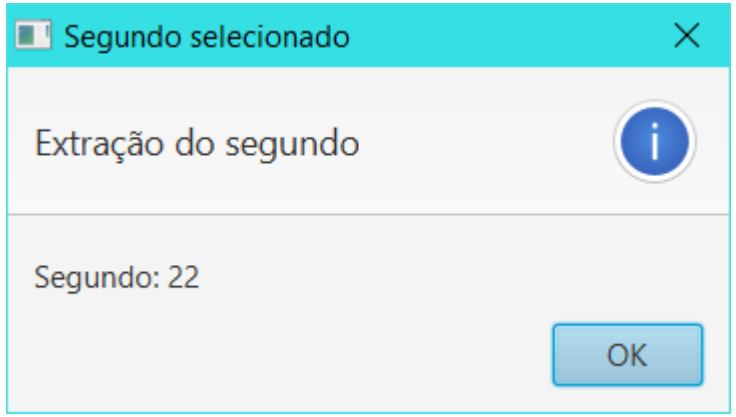
```
266    @FXML public void minuto() {
267
268        // Obtém a hora, minuto e segundo lidos a partir dos controles Spinner spHoraAtual, spMinutoAtual e spSegundoAtual
269        // formatados com dois dígitos
270        String horaAtual = String.format("%02d", spHoraAtual.getValue());
271        String minutoAtual = String.format("%02d", spMinutoAtual.getValue());
272        String segundoAtual = String.format("%02d", spSegundoAtual.getValue());
273        // Monta uma String com o horário completo
274        String horarioAtualString = horaAtual + ":" + minutoAtual + ":" + segundoAtual;
275
276        // Converte o horário de String para LocalTime no formato HH:mm:ss
277        DateTimeFormatter formatador = DateTimeFormatter.ofPattern("HH:mm:ss");
278        LocalTime horarioAtualFormatado = LocalTime.parse(horarioAtualString, formatador);
279
280        Alert alert = new Alert(AlertType.INFORMATION);
281        alert.setTitle("Minuto selecionado");
282        alert.setHeaderText("Extração do minuto");
283        // Apresenta somente os minutos formatado com dois dígitos
284        alert.setContentText( "Minuto: " + String.valueOf( String.format("%02d", horarioAtualFormatado.getMinute() )));
285        alert.showAndWait();
286    }
```

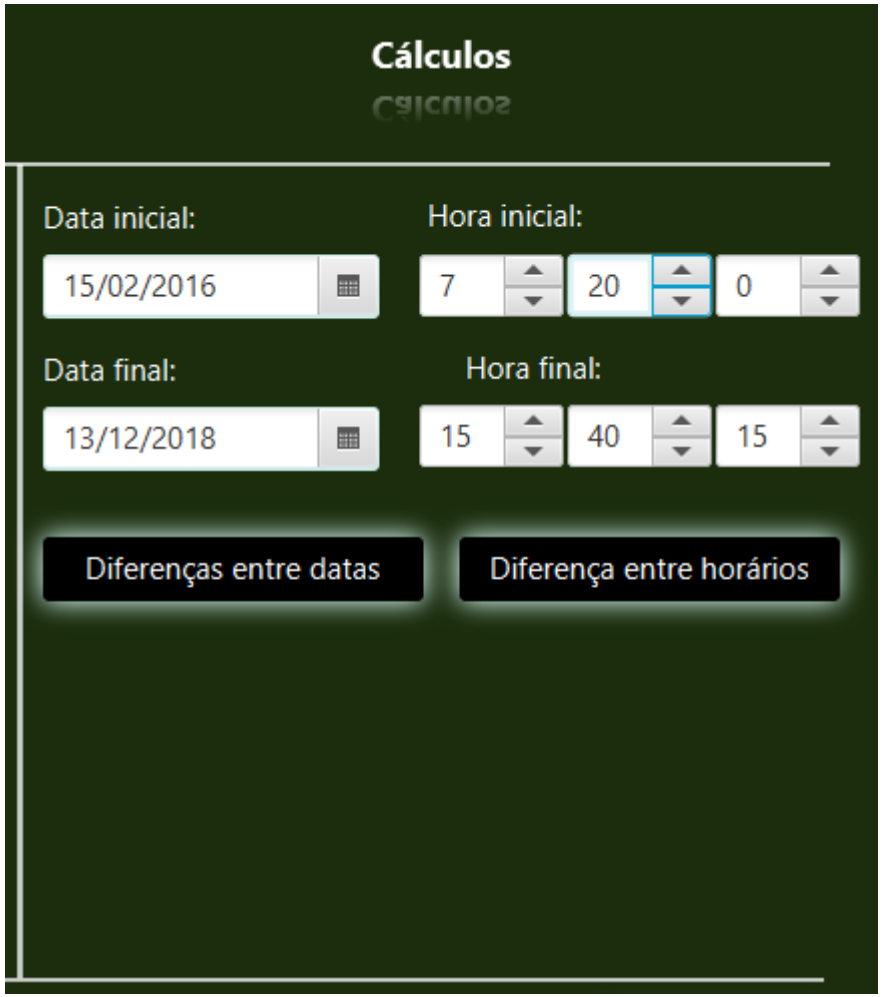


OBTER O SEGUNDO FORMATADO (LOCALTIME)

JFXDataHoraControle.java

```
289    @FXML public void segundo() {
290
291        // Obtém a hora, minuto e segundo lidos a partir dos controles Spinner spHoraAtual, spMinutoAtual e spSegundoAtual
292        // formatados com dois dígitos
293        String horaAtual = String.format("%02d", spHoraAtual.getValue());
294        String minutoAtual = String.format("%02d", spMinutoAtual.getValue());
295        String segundoAtual = String.format("%02d", spSegundoAtual.getValue());
296        // Monta uma String com o horário completo
297        String horarioAtualString = horaAtual + ":" + minutoAtual + ":" + segundoAtual;
298
299        // Converte o horário de String para LocalTime no formato HH:mm:ss
300        DateTimeFormatter formatador = DateTimeFormatter.ofPattern("HH:mm:ss");
301        LocalTime horarioAtualFormatado = LocalTime.parse(horarioAtualString, formatador);
302
303        Alert alert = new Alert(AlertType.INFORMATION);
304        alert.setTitle("Segundo selecionado");
305        alert.setHeaderText("Extração do segundo");
306        // Apresenta somente os segundos formatado com dois dígitos
307        alert.setContentText( "Segundo: " + String.valueOf( String.format("%02d", horarioAtualFormatado.getSecond() )));
308        alert.showAndWait();
309    }
```

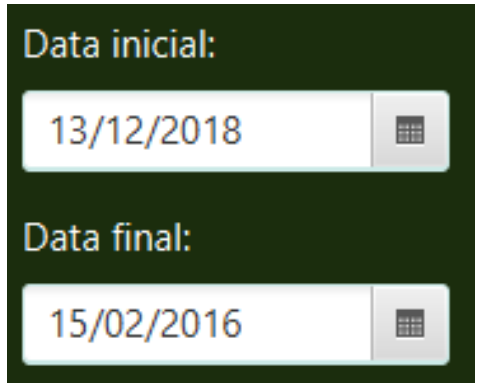


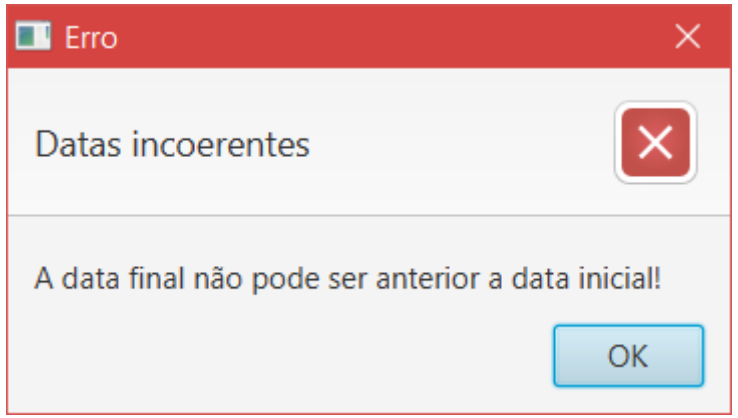

Cálculos
Data inicial:
15/02/2016
Hora inicial:
7
20
0
Data final:
13/12/2018
Hora final:
15
40
15
Diferenças entre datas
Diferença entre horários

DIFERENÇA ENTRE DATAS (PERIOD E DURATION)

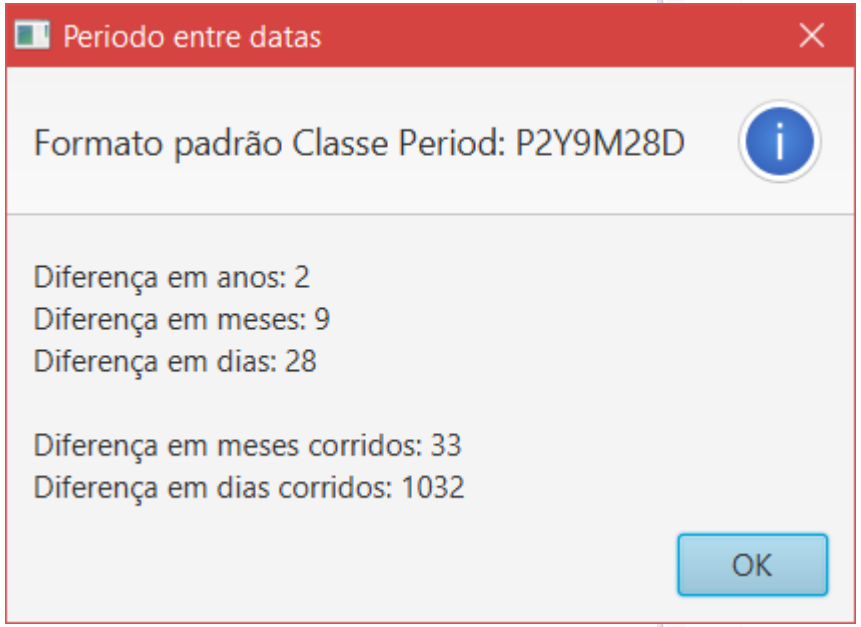
JFXDataHoraControle.java

```
312     @FXML public void diferencaDatas() {
313
314         // Verifica se as datas selecionadas como inicio e fim estão coerentes
315         if (dpDataFinal.getValue().isBefore(dpDataInicial.getValue())) {
316
317             Alert alert = new Alert(AlertType.ERROR);
318             alert.setTitle("Erro");
319             alert.setHeaderText("Datas incoerentes");
320             alert.setContentText( "A data final não pode ser anterior a data inicial!");
321             alert.showAndWait();
322         }else{
323
324             // Calcula e apresenta a diferença entre as datas
325         }
326     }
```





PERIOD DURATION



JFXDataHoraControle.java

```
324         // Obtém a data inicial e a data final lidas a partir dos controles DatePicker dpDataIncial e dpDataFinal
325         // convertendo para LocalDate.
326         // Esses objetos serão utilizados com a classe Period
327         LocalDate dataInicial = LocalDate.parse(dpDataInicial.getValue().toString());
328         LocalDate dataFinal = LocalDate.parse(dpDataFinal.getValue().toString());
329
330         // Obtém separadamente o ano, mês e dia inical e final para utilizá-los em um objeto LocalDateTime
331         int anoInicial = dataInicial.getYear();
332         int mesInicial = dataInicial.getMonthValue();
333         int diaInicial = dataInicial.getDayOfMonth();
334         int anoFinal = dataFinal.getYear();
335         int mesFinal = dataFinal.getMonthValue();
            int diaFinal = dataFinal.getDayOfMonth();

            // Cria objetos LocalDateTime a partir do ano, mês e dia (selecionados nos DatePickers) e
            // zeros para hora, minuto e segundos (uma vez que controles DatePickers não selecionam horários).
            // Esses objetos serão utilizados com a classe Duration
            LocalDateTime dataHoraInicial = LocalDateTime.of(anoInicial, mesInicial, diaInicial, 0,0,0);
            LocalDateTime dataHoraFinal = LocalDateTime.of(anoFinal, mesFinal, diaFinal, 0,0,0);

            // Calcula o período entre as datas (a partir de referências LocalDate)
            // Para extração dos anos, meses e dias
            Period periodoData = Period.between(dataInicial, dataFinal);

            // Calcula a duração entre as datas (a partir de referências LocalDateTime)
            // Para extração dos dias corridos
            Duration duracaoData = Duration.between(dataHoraInicial, dataHoraFinal);

            Alert alert = new Alert(AlertType.INFORMATION);
            alert.setTitle("Periodo entre datas");
            alert.setHeaderText("Formato padrão Classe Period: " + periodoData);
            alert.setContentText( "Diferença em anos: " + periodoData.getYears() +
                                  "\nDiferença em meses: " + periodoData.getMonths() +
                                  "\nDiferença em dias: " + periodoData.getDays() +

                                  "\n\nDiferença em meses corridos: " + periodoData.toTotalMonths() +
                                  "\nDiferença em dias corridos: " + duracaoData.toDays());
356
357
358
359
360
361
            alert.showAndWait();
```

DIFERENÇA ENTRE HORÁRIOS (LOCALTIME E DURATION)

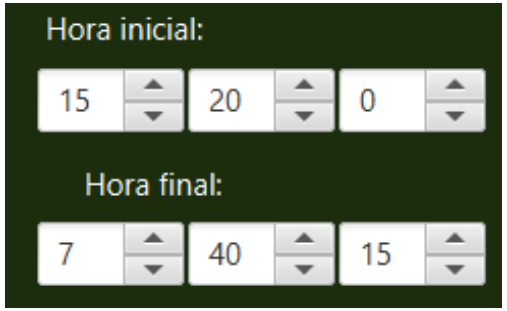
JFXDataHoraControle.java

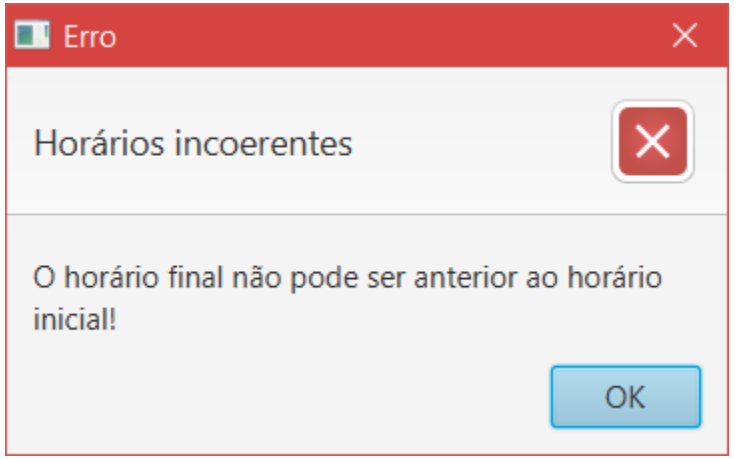
```
366     @FXML public void diferencaHorarios() {
367
368         // Obtém a hora, minuto e segundo inicial lidos a partir dos controles Spinner spHoraInicial, spMinutoInicial e spSegundoInicial
369         // formatados com dois dígitos
370         String horaInicial = String.format("%02d", spHoraInicial.getValue());
371         String minutoInicial = String.format("%02d", spMinutoInicial.getValue());
372         String segundoInicial = String.format("%02d", spSegundoInicial.getValue());
373         // Monta uma String com o horário inicial completo
374         String horarioInicialString = horaInicial + ":" + minutoInicial + ":" + segundoInicial;
375
376         // Obtém a hora, minuto e segundo final lidos a partir dos controles Spinner spHoraFinal, spMinutoFinal e spSegundoFinal
377         // formatados com dois dígitos
378         String horaFinal = String.format("%02d", spHoraFinal.getValue());
379         String minutoFinal = String.format("%02d", spMinutoFinal.getValue());
380         String segundoFinal = String.format("%02d", spSegundoFinal.getValue());
381         // Monta uma String com o horário final completo
382         String horarioFinalString = horaFinal + ":" + minutoFinal + ":" + segundoFinal;
383
384         // Converte as Strings horarioInicialString e horarioFinalString para LocalTime
385         LocalTime horarioInicial = LocalTime.parse(horarioInicialString);
386         LocalTime horarioFinal = LocalTime.parse(horarioFinalString);
```

DIFERENÇA ENTRE HORÁRIOS (DURATION)

JFXDataHoraControle.java

```
388         // Verifica se os horários selecionados como inicio e fim estão coerentes
389         if (horarioFinal.isBefore(horarioInicial)) {
390 
391             Alert alert = new Alert(AlertType.ERROR);
392             alert.setTitle("Erro");
393             alert.setHeaderText("Horários incoerentes");
394             alert.setContentText( "O horário final não pode ser anterior ao horário inicial!");
395             alert.showAndWait();
396         }else{
397 
398             // Calcula e apresenta a diferença entre os horários
399         }
400     }
```

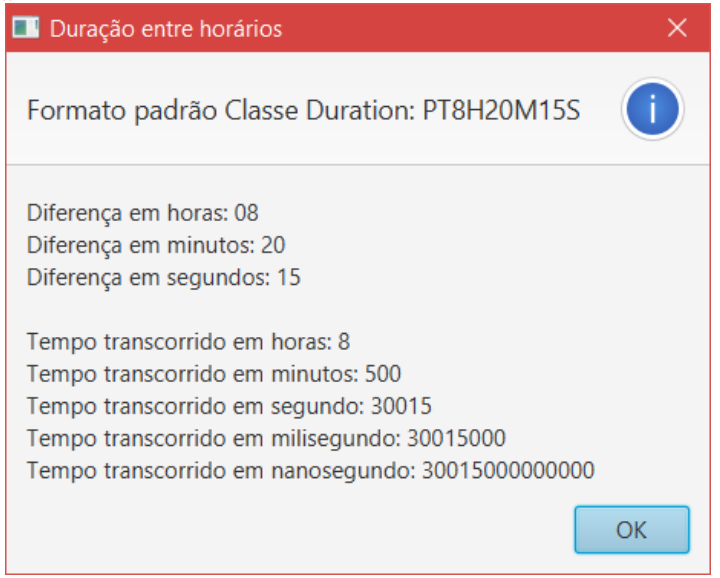




DURATION

JFXDataHoraControle.java

```java
// Calcula a duração entre os horários (a partir de referências LocalTime)
Duration duracaoHorario = Duration.between(horarioInicial, horarioFinal);

// O método between da classe Duration calcula a diferença entre os horários em segundos.
// A extração das horas, minutos, milisegundos e nanosegundos transcorridos é feita por métodos, mas,
// a diferença em "hora relógio" deve ser calculada...

// A partir da duração em segundos
long duracaoHorarioSegundosTotal = duracaoHorario.getSeconds();
// Dividindo por 3600 (1 hora = 60 minutos, 1 minuto = 60 segundos, 60 * 60 = 3600) obtém-se a quantidade de horas
long duracaoHorarioHoras = duracaoHorarioSegundosTotal / 3600;
// Se nessa divisão houver resto, basta dividir esse resto por 60 para obter a quantidade de minutos
long duracaoHorarioMinutos = (duracaoHorarioSegundosTotal % 3600) / 60;
// Os segundos são obtidos somando as horas e minutos (convertidos novamente para segundos) e subtraindo do total de segundos
long duracaoHorarioSegundos = duracaoHorarioSegundosTotal - ((duracaoHorarioHoras * 3600) + (duracaoHorarioMinutos * 60));

// Tendo os valores da hora, minuto e segundos basta converte-los para String formatados para dois dígitos
String duracaoHorarioHorasString = String.format("%02d", duracaoHorarioHoras);
String duracaoHorarioMinutosString = String.format("%02d", duracaoHorarioMinutos);
String duracaoHorarioSegundosString = String.format("%02d", duracaoHorarioSegundos);

Alert alert = new Alert(AlertType.INFORMATION);
alert.setTitle("Duração entre horários");
alert.setHeaderText("Formato padrão Classe Duration: " + duracaoHorario);
alert.setContentText(
                        "Diferença em horas: " + duracaoHorarioHorasString +
                        "\nDiferença em minutos: " + duracaoHorarioMinutosString +
                        "\nDiferença em segundos: " + duracaoHorarioSegundosString +

                        "\n\nTempo transcorrido em horas: " + duracaoHorario.toHours() +
                        "\nTempo transcorrido em minutos: " + duracaoHorario.toMinutes() +
                        "\nTempo transcorrido em segundo: " + duracaoHorario.getSeconds() +
                        "\nTempo transcorrido em milisegundo: " + duracaoHorario.toMillis() +
                        "\nTempo transcorrido em nanosegundo: " + duracaoHorario.toNanos());
alert.showAndWait();
```



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...

CONTEXTUALIZAÇÃO – MATRIX

- 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.