Datas em Java

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

Herculano De Biasi
herculano.debiasi@unoesc.edu.br

Tópicos

- Bibliografia
- Introdução
- Classe `java.util.Date`
- Classes `java.sql.Date/java.sql.Time/java.sql.TimeStamp`
- Classe `Calendar`
- Classes `DateFormat` e `SimpleDateFormat`
- Padrão ISO 8601
- Classe `Instant`
- Classes `LocalDate` e `LocalDateTime`
- Conversões
- Contas com datas
- Classes `Period` e `Duration`

UNOESC

BIBLIOGRAFIA

- SILVEIRA, Paulo; TURINI, Rodrigo. **Java 8 Prático**: *Lambdas*, *Streams* e os novos recursos da linguagem. Casa do Código, São Paulo: 2014.

Apoiadores:

UNOESC

INTRODUÇÃO

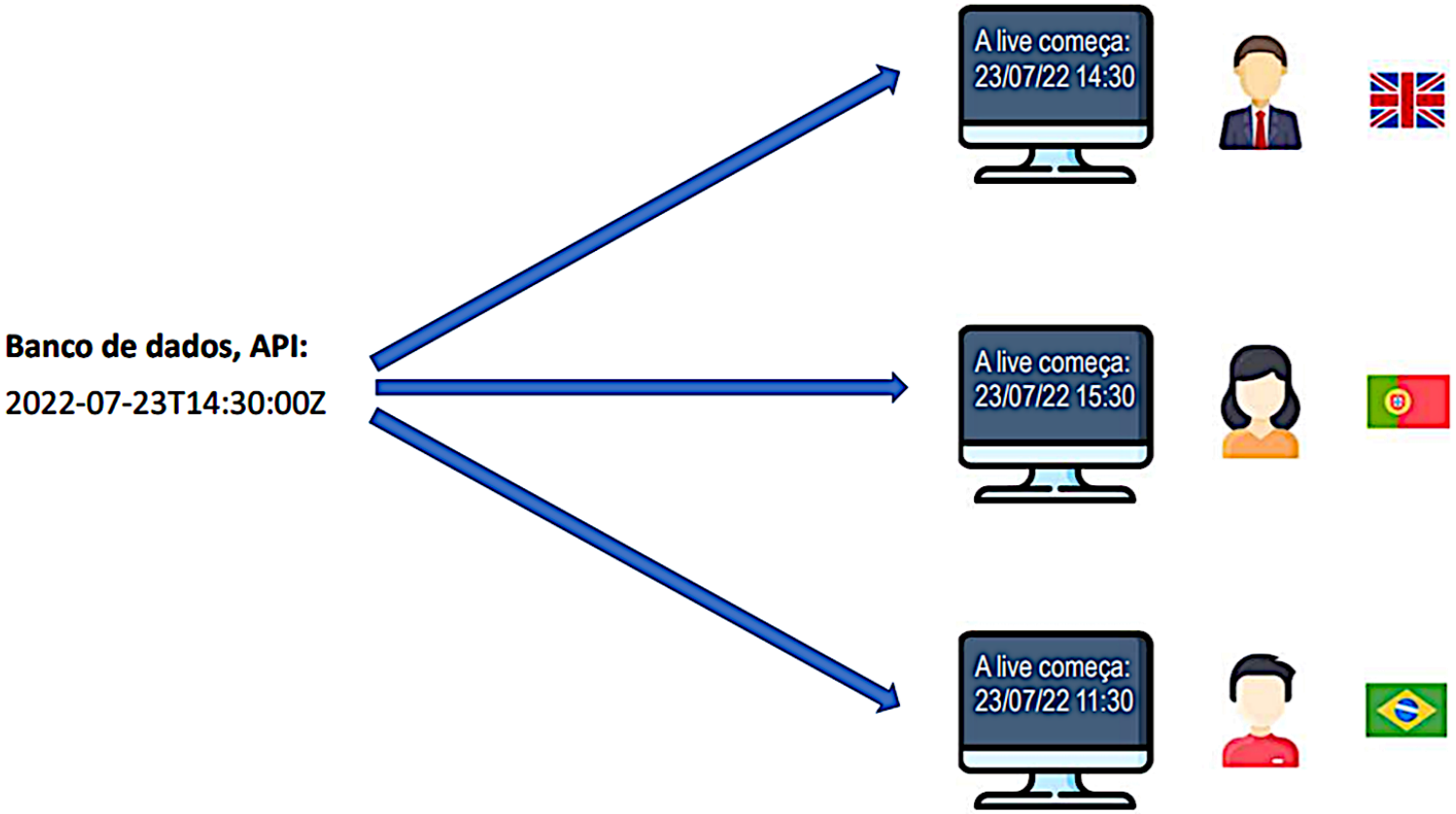
- Data-[hora] local: Utilizado quando o momento exato não interessa a pessoas de outro fuso horário sendo mais usados em sistemas locais, de região única
  - ano-mês-dia-[hora] sem fuso horário
  - [hora] opcional

- Data-hora global: Utilizado quando o momento exato interessa a pessoas de outro fuso horário sendo comuns em sistemas multirregião e sistemas *web*
  - ano-mês-dia-hora com fuso horário

- Duração: tempo decorrido entre duas data-horas

INTRODUÇÃO

- Fusos horários: As boas práticas recomendam armazenar UTC e mostrar local



Apoiadores:

UNOESC

Classe Java.Util.Date

- A classe `java.util.Date` representa um instante de tempo que armazena
  - O número de milissegundos desde a meia-noite do dia 1° de janeiro de 1970 GMT (UTC)
    - GMT (*Greenwich Mean Time*): Fuso horário do meridiano de Greenwich
    - UTC (*Coordinated Universal Time*)
  - A hora oficial do Brasil é 3 horas a menos que a de GMT, sendo expressa como GMT-3

- As classes `Date` e `Calendar` eram as formas mais usadas de trabalhar com datas no Java até a versão 7; o Java 8 introduziu uma nova APIs de datas
  - A maioria dos métodos da classe `Date` estão classificados como *deprecated* (depreciados), ou seja, são métodos que não devem ser mais utilizados
  - Eles começaram a ser substituídos com o surgimento da classe `Calendar`, para haver suporte correto à internacionalização do sistema de datas

Classe Java.Util.Date

- Principais métodos
  - `Date()`: Construtor passando o número de milissegundos decorridos desde 01.01.1970
  - `Date.from()`: Método estático que recebe um objeto da classe `Instant`
  - `Date.parse()`: Método estático que recebe uma *string* no formato ISO 8601

Classe Java.Util.Date

- Exemplo

```
1  package datasantigas;
2
3  import java.text.DateFormat;
4  import java.text.SimpleDateFormat;
5  import java.util.Date;
6
7  public class TestaDate {
8
9      public static void main(String[] args)  {
10         Date dataHoraAtuais = new Date();
11
12         System.out.println("Data/hora atuais: " + dataHoraAtuais);
13
14         System.out.println();
15         System.out.println(new Date(0L));                    // 31 dezembro de 1969, 21h
16         System.out.println(new Date(1000L * 60 * 60 * 3));   // 1o. janeiro de 1970, meia-noite
17
18         System.out.println();
19         Date dataHora1 = new Date(85, 7, 17, 17, 45, 0);     // Janeiro = 0, agosto = 7
20         Date dataHora2 = new Date("08/17/85 05:45:00 PM");
21         Date dataHora3 = new Date("08/17/985 05:45:00 PM");
22         Date dataHora4 = new Date("08/17/1985 05:45:00 PM");
23         Date dataHora5 = new Date(1950, 6, 16, 12, 0, 0);    // Julho = 6
24         System.out.println(dataHora1);
25         System.out.println(dataHora2);
26         System.out.println(dataHora3);
27         System.out.println(dataHora4);
28         System.out.println(dataHora5);
29
30         System.out.println();
31         DateFormat df = new SimpleDateFormat("d MMM yyyy G, HH:mm:ss.S Z");
32         System.out.println(df.format(new Date(Long.MIN_VALUE)));
33         System.out.println(df.format(new Date(1000L * 60 * 60 * 3)));
34         System.out.println(df.format(new Date(Long.MAX_VALUE)));
35     }
36
37 }
```

UNOESC

Classes java.sql.Date/Time/TimeStamp

- A classe `java.sql.Date` é um *wrapper* em volta de um valor em milissegundos que permite que o JDBC identifique-o como um valor SQL do tipo `DATE`
  - Esta classe herda de `java.util.Date`
  - Não armazena informações relativas a fuso horários

- As classes `java.sql.Time` e `java.sql.Timestamp` mapeiam respectivamente para os tipos `TIME` e `TIMESTAMP` do SQL

Classes java.sql.Date/Time/TimeStamp

- Principais métodos
  - `Date(long milissegundos):`
  - `setTime(long milissegundos):`
  - `toString():`
  - `valueOf(LocalDate)/valueOf(String):`
  - `Instant toInstant()`: Converte para objeto do tipo `Instant`
  - `LocalDate toLocalDate()`: Converte para objeto do tipo `LocalDate`

Classes Java.Sq

- Exemplos

```java
package sql;

import java.sql.Date;
import java.text.SimpleDateFormat;
import java.time.Instant;
import java.time.LocalDate;
import java.time.Month;
import java.util.Calendar;

public class DatasSQL {

    public static void main(String[] args) {
        long now = System.currentTimeMillis();
        java.sql.Date sqlDate = new Date(now);

        System.out.println("currentTimeMillis: " + now);
        System.out.println("SqlDate.....: " + sqlDate);
        System.out.println("SqlDate.getTime(): " + sqlDate.getTime());

        System.out.println();

        java.sql.Date sqlDate1 = Date.valueOf("1975-12-25");
        System.out.println("SqlDate1: " + sqlDate1);

        java.sql.Date sqlDate2 = Date.valueOf(LocalDate.of(1975, Month.DECEMBER, 25));
        System.out.println("SqlDate2: " + sqlDate2);

        java.util.Date dataUtil = new java.util.Date();
        System.out.println("\nData do pacote util: " + dataUtil);

        // Convertendo de java.util.Date para java.sql.Date
        java.sql.Date dataSQL1 = new java.sql.Date(dataUtil.getTime());
        System.out.println("Data do pacote SQL.: " + dataSQL1);

        // Convertendo de java.util.Date para java.sql.Date com SimpleDataFormat() e valueOf()
        java.util.Date data = new java.util.Date();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        String fd = sdf.format(data);
        java.sql.Date dataSQL2 = java.sql.Date.valueOf(fd);
        System.out.println("Data SQL.....: " + dataSQL2);

        // Convertendo para Instant
        Calendar cln = Calendar.getInstance();
        cln.set(Calendar.MONTH, 07);
        cln.set(Calendar.DATE, 12);
        cln.set(Calendar.YEAR, 2021);
        java.util.Date d = cln.getTime();
        Instant inst = d.toInstant();
        System.out.println("\nData original.: " + d.toString());
        System.out.println("Objeto Instant: " + inst);
    }

}
```

UNOESC

Classe Calendar

- A classe `java.util.Calendar` é muito utilizada pois fornece suporte correto à internacionalização do sistema de datas
  - É uma classe abstrata, não pode ser instanciada, portanto para obter um calendário é necessário usar o método estático `getInstance()`
  - Armazena a data no formato de milissegundos desde 01.01.1970 como `Date`
  - Suporta 'campos de calendário', como ano, mês, dia do mês, etc
  - Meses são representados por constantes, sendo que janeiro (`JANUARY`) tem o valor 0
  - Suporta operações básicas de adição de datas/horas

- Principais métodos
  - `get()/set()`: Recupera/define um campo de uma data
  - `getTime()/setTime()`: Recupera/define uma data usando um objeto do tipo `Date`
  - `add()`: Soma/subtrai um valor em um campo de data

UNOESC

Classe Calendar

- Exemplo

```java
package datasantigas;

import java.time.Instant;
import java.util.Calendar;
import java.util.Date;

public class TestaCalendario {

    public static void main(String[] args) {
        Calendar c = Calendar.getInstance();

        c.set(Calendar.YEAR, 1985);
        c.set(Calendar.MONTH, Calendar.AUGUST);
        c.set(Calendar.DAY_OF_MONTH, 17);

        System.out.println("Data e hora atuais: " + c.getTime());
        System.out.println("Ano.............: " + c.get(Calendar.YEAR));
        System.out.println("Mês (janeiro=0)...: " + c.get(Calendar.MONTH));
        System.out.println("Dia do mês.......: " + c.get(Calendar.DAY_OF_MONTH));

        System.out.println("\nHora do dia...: " + c.get(Calendar.HOUR_OF_DAY));
        System.out.println("Minuto do dia.: " + c.get(Calendar.MINUTE));
        System.out.println("Segundo do dia: " + c.get(Calendar.SECOND));

        c.add(Calendar.MONTH, 1);
        c.add(Calendar.DAY_OF_MONTH, -1);
        c.add(Calendar.HOUR_OF_DAY, 5);
        System.out.println("\nMês (janeiro=0): " + c.get(Calendar.MONTH));
        System.out.println("Dia do mês.....: " + c.get(Calendar.DAY_OF_MONTH));
        System.out.println("Hora do dia....: " + c.get(Calendar.HOUR_OF_DAY));

        Date data = Date.from(Instant.parse("1985-08-17T17:42:06Z"));
        c.setTime(data);
        System.out.println(data);
    }

}
```

UNOESC

Classes DateFormat e SimpleDateFormat

- As classes `java.text.DateFormat` e `java.text.SimpleDateFormat` trabalham com formatos de conversão entre os tipos `Date` e `String`, tanto em datas quanto em datas/horas
  - `dd/MM/yyyy` → 17/08/1985
  - `dd/MM/yyyy HH:mm:ss` → 17/08/1985 17:42:06

- Observações
  - `DateFormat` trabalha com constantes como `FULL`, `LONG`, `MEDIUM` e `SHORT` que indicam estilos já predefinidos de formatação
  - `SimpleDateFormat` permite uma forma mais flexível/poderosa de formatação
  - As duas classes suportam internacionalização (localização) e definição de fusos horários

Classes DateFormat e SimpleDateFormat

- Principais métodos
  - Construtores: Aceitam uma *string* que descreve a data/hora de forma personalizada
  - `parse()`: Recebe uma *string* e retorna um objeto `Date` ou `DateTime`
  - `format()`: Recebe um objeto do tipo Date e retorna uma *string*
  - `setTimeZone()`: Define um fuso horário a partir de um objeto `TimeZone`
  - `getDateInstance(estilo, locale)`: Retorna um formatador

Classes DateFormat e SimpleDateFormat

- Exemplo

```java
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.Instant;
import java.util.Date;
import java.util.Locale;
import java.util.TimeZone;

public class TestaDatesFormatadas {

    public static void main(String[] args) throws ParseException {
        SimpleDateFormat sdf1 = new SimpleDateFormat("dd/MM/yyyy");
        SimpleDateFormat sdf2 = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

        Date d1 = sdf1.parse("17/08/1985");
        Date d2 = sdf2.parse("17/08/1985 17:42:06");

        System.out.println("d1 padrão...: " + d1);
        System.out.println("d2 padrão...: " + d2);

        System.out.println("d1 formatado: " + sdf1.format(d1));
        System.out.println("d2 formatado: " + sdf2.format(d2));
        System.out.println();

        SimpleDateFormat sdfGMT = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
        sdfGMT.setTimeZone(TimeZone.getTimeZone("GMT"));

        System.out.println(sdfGMT.format(new Date(0L)));                 // 31 dezembro de 1969, 21h
        System.out.println(sdfGMT.format(new Date(1000L * 60 * 60 * 3))); // 1o. janeiro de 1970, 5 da manhã
        System.out.println();

        Date data = Date.from(Instant.parse("1985-08-17T17:42:06Z"));
        System.out.println(data);
        System.out.println(sdfGMT.format(data));

        Locale localBR = new Locale("pt", "BR");
        DateFormat df = DateFormat.getDateInstance(DateFormat.FULL, localBR);
        System.out.println(df.format(new Date()));
    }

}
```

UNOESC

Padrão ISO 8601

- O padrão ISO 8601 define um padrão para representação de datas e horas no formato `yyyy-MM=ddTHH:mm:ssZ`
  - O `Z` indica que é um horário em UTC e por causa disso também é chamado de 'Zulu *time*'

- Outros formatos para data e hora são possíveis
  - Locais

```
2022-07-21
2022-07-21T14:52
2022-07-22T14:52:09
2022-07-22T14:52:09.4073
```

  - Globais
    - Ponto representa frações de segundo
    - Sinal de – representa fuso horário

```
2022-07-23T14:52:09Z
2022-07-23T14:52:09.254935Z
2022-07-23T14:52:09-03:00
```

Classe Instant

- A classe `Instant` trabalha com data global, ou seja, consegue representar um momento (data e hora) no tempo em formato UTC e consegue converter de *strings* em UTC para um objeto da classe `Date`

- Principais métodos
  - `now()`: Cria uma data/hora global no fuso de GMT
  - `parse(string)`: *String* deve estar no formato ISO 8601
  - Exemplo: `Date data = Date.from(Instant.parse("1985-08-17T17:42:06Z"));`

Classe Instant

- Exemplo

```java
package datasnovas;

import java.time.Instant;

public class TesteInstant {

    public static void main(String[] args) {
        Instant insGMT1 = Instant.now();
        Instant insGMT2 = Instant.parse("1985-08-17T20:00:00Z");      // Especifica horário GMT
        Instant insGMT3 = Instant.parse("1985-08-17T20:00:00-03:00"); // Formato ISO 8601 com fuso diferente

        System.out.println("Data/hora global: " + insGMT1);
        System.out.println("Data/hora global: " + insGMT2);
        System.out.println("Data/hora global: " + insGMT3);
    }

}
```

Classes LocalDate, LocalTime e LocalDateTime

- `LocalDate`, `LocalTime` e `LocalDateTime` trabalham, respectivamente, com datas locais, horas locais e datas/horas locais

- Pode-se usar a classe `DateTimeFormatter` com o método estático `ofPattern()`, com uma *string* de controle, para formatação de datas e horas

- Principais métodos
  - `now()`: Cria uma data ou data/hora local
  - `LocalDate.of(ano, mês, dia)`: Cria uma data local
  - `LocalTime.of(hora, minuto, segundo)`: Cria uma hora local
  - `LocalDateTime.of(ano, mês, dia, hora, minuto, segundo)`: Cria data/hora local
  - `getHour()/getMinute()/getSecond()`: Retorna as horas, minutos e segundos
  - `getDayOfMonth()`: Retorna o dia do mês (1 a 31)
  - `getMonth().getValue()`: Retorna o mês (1 a 12)
  - `getYear()`: Retorna o ano

UNOESC

Classes LocalDate, LocalTime e LocalDateTime

- Exemplo

```java
package datasnovas;

import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;

public class NovasDatasHorasLocais {

    public static void main(String[] args) {
        LocalDate ld1 = LocalDate.now();
        LocalDateTime ldt1 = LocalDateTime.now();
        System.out.println("Data local.....: " + ld1);
        System.out.println("Data/hora local.: " + ldt1);
        System.out.println();

        LocalDate ld2 = LocalDate.of(1985, 8, 17);
        LocalDateTime ldt2 = LocalDateTime.of(1985, 8, 17, 17, 45, 0);
        System.out.println("Data local.....: " + ld2);
        System.out.println("Data/hora local....: " + ldt2);
        System.out.printf("Dia/mês/ano.....: %02d/%02d/%04d\n", ldt2.getDayOfMonth(),
                                                                ldt2.getMonth().getValue(),
                                                                ldt2.getYear());
        System.out.printf("Hora:minuto:segundo: %02d:%02d:%02d\n", ldt2.getHour(),
                                                                   ldt2.getMinute(),
                                                                   ldt2.getSecond());

        DateTimeFormatter fmt1 = DateTimeFormatter.ofPattern("dd/MM/yyyy");
        LocalDate ldFmt1 = LocalDate.parse("17/08/1985", fmt1);
        System.out.println("\nldFmt1 padrão...: " + ldFmt1);
        System.out.println("ldFmt1 formatada: " + ldFmt1.format(fmt1));

        DateTimeFormatter fmt2 = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");
        LocalDateTime ldFmt2 = LocalDateTime.parse("17/08/1985 17:45", fmt2);
        System.out.println("\nldFmt2 padrão...: " + ldFmt2);
        System.out.println("ldFmt2 formatada: " + ldFmt2.format(fmt2));

        DateTimeFormatter fmt3 = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm")
                                 .withZone(ZoneId.systemDefault());
        Instant insFmt3 = Instant.parse("1985-08-17T17:45:00Z");
        System.out.println("\nInstant padrão...: " + insFmt3);
        System.out.println("Instant formatada: " + fmt3.format(insFmt3));
    }

}
```

UNOESC

CONVERSÕES

- Convertendo de data/hora global para local

```java
package datasnovas;

import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZoneId;

public class Conversoes {
    public static void main(String[] args) {
        Instant insGlobal = Instant.parse("1985-08-17T01:45:00Z");

//      for (String s : ZoneId.getAvailableZoneIds()) {
//          System.out.println(s);
//      }

        // Converter de data global para data local
        LocalDate r1 = LocalDate.ofInstant(insGlobal, ZoneId.systemDefault());
        LocalDate r2 = LocalDate.ofInstant(insGlobal, ZoneId.of("Portugal"));
        LocalDateTime r3 = LocalDateTime.ofInstant(insGlobal, ZoneId.systemDefault());
        LocalDateTime r4 = LocalDateTime.ofInstant(insGlobal, ZoneId.of("Portugal"));

        System.out.println("r1: " + r1);
        System.out.println("r2: " + r2);

        System.out.println("r3: " + r3);
        System.out.println("r4: " + r4);
    }
}
```

UNOESC

Contas com Datas

- As novas classes suportam operações de cálculo entre as datas com os métodos
  - `minusHours()`,`minusDays()`,`minusMonths()`,`minusYears()`,etc subtraem datas/horas
  - `plusHours()`,`plusDays()`,`plusMonths()`,`plusYears()`,etc adiciona datas/horas

```java
package datasnovas;

import java.time.LocalDate;
import java.time.LocalDateTime;

public class ContasComDatas {

    public static void main(String[] args) {
        LocalDate ld = LocalDate.parse("1985-08-17");
        LocalDate semanaPassada = ld.minusDays(7);
        LocalDate semanaQueVem = ld.plusDays(7);

        System.out.println(semanaPassada);
        System.out.println(ld);
        System.out.println(semanaQueVem);
        //-------------
        LocalDateTime ldt = LocalDateTime.parse("1985-08-17T17:45:00");
        LocalDateTime horaAnoPassado = ldt.minusHours(1).minusYears(1);
        LocalDateTime horaAnoQueVem = ldt.plusHours(1).plusYears(1);

        System.out.println(horaAnoPassado);
        System.out.println(ldt);
        System.out.println(horaAnoQueVem);
    }

}
```

UNOESC

Contas com Datas

- No caso da classe `Instant` só existem os métodos `minus()` e `plus()` que devem receber, além disso, uma unidade temporal (no exemplo se usa a classe `ChronoUnit` para isso)

```java
package datasnovas;

import java.time.Instant;

public class ContasComChrono {

    public static void main(String[] args) {
        Instant hoje = Instant.parse("1985-08-17T01:45:00Z");
        Instant ontem = hoje.minus(1, ChronoUnit.DAYS);
        Instant amanha = hoje.plus(1, ChronoUnit.DAYS);

        System.out.println(ontem);
        System.out.println(hoje);
        System.out.println(amanha);

        LocalDate independencia = LocalDate.of(2022, 9, 7);
        LocalDate republica = LocalDate.of(2022, 11, 15);

        long dias = ChronoUnit.DAYS.between(independencia, republica);
        System.out.printf("\nSão %s dias de diferença.", dias);

        long meses = ChronoUnit.MONTHS.between(independencia, republica);
        System.out.printf("\nSão %s meses de diferença.", meses);
    }

}
```

UNOESC

Contas com Datas

- Outros métodos úteis das classes `LocalDate` e `LocalDateTime` são
  - `isLeapYear()`:Testa se é ano bissexto
  - `isBefore()/isAfter()`:Testa se vem antes/depois
  - `isEqual/equals()`:Testa se é igual
  - `lengthOfMonth()/lengthOfYear()`: Número de dias do mês/ano

```
3  import java.time.LocalDate;
4  import java.time.LocalDateTime;
5  import java.time.LocalTime;
6
7  public class ContasComDatasHoras {
8
9      public static void main(String[] args) {
10         LocalDate data = LocalDate.now();
11         System.out.println("Ano bissexto.........: " + (data.isLeapYear() ? "Sim" : "Não"));
12         System.out.println("Número de dias do mês: " + data.lengthOfMonth());
13         System.out.println("Número de dias do ano: " + data.lengthOfYear());
14
15         LocalDate anoNovo = LocalDate.of(2022, 1, 1);
16         LocalDate natal = LocalDate.of(2022, 12, 25);
17
18         System.out.println("\nAno novo antes do Natal? " + (anoNovo.isBefore(natal) ? "Sim" : "Não"));
19         System.out.println("Ano novo depois do Natal? " + (anoNovo.isAfter(natal) ? "Sim" : "Não"));
20         System.out.println("Ano novo na mesma data do Natal? " + (anoNovo.isEqual(natal) ? "Sim" : "Não"));
21
22         LocalTime almoco = LocalTime.of(12, 0);
23         LocalTime jantar = LocalTime.of(20, 30);
24
25         LocalDateTime anoNovoAlmoco = LocalDateTime.of(anoNovo, almoco);
26         LocalDateTime natalJantar = LocalDateTime.of(natal, jantar);
27
28         System.out.println("\nAlmoço antes da janta? " + (anoNovoAlmoco.isBefore(natalJantar) ? "Sim" : "Não"));
29         System.out.println("Almoço antes da janta? " + (anoNovoAlmoco.isAfter(natalJantar) ? "Sim" : "Não"));
30         System.out.println("Almoço na mesma hora da janta? " + (anoNovoAlmoco.equals(natalJantar) ? "Sim" : "Não"));
31     }
32
33 }
```

Classes Period e Duration

- Classes
  - `Period`: Representa uma quantidade de tempo em anos, meses e dias.
  - `Duration`: Representa uma quantidade de tempo em dias, horas, minutos, segundos e nanosegundos

- Métodos mais importantes
  - `minus/plus`: Retorna uma cópia do instante subtraído/adicionado do tempo informado
  - `with`: Retorna uma cópia da data com o ajuste solicitado

Classes Period e Duration

Exemplo de `Period`

```java
package datasnovas;

import java.time.DayOfWeek;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.Period;
import java.time.temporal.TemporalAdjusters;

public class Periodo {

    public static void main(String[] args) {
        LocalDateTime inicio = LocalDateTime.of(2017, 1, 25, 10, 00, 00);
        LocalDateTime termino = LocalDateTime.of(2017, 12, 26, 11, 20, 15);
        Period periodo = Period.between(inicio.toLocalDate(), termino.toLocalDate());

        System.out.printf("%s anos, %s meses e %s dias\n", periodo.getYears(), periodo.getMonths(), periodo.getDays());
        // ------------------
        LocalDate flamengoBrasileiro = LocalDate.of(2009, 12, 6);
        LocalDate flamengoCopaDoBrasil = LocalDate.of(2013, 11, 27);
        Period dif = Period.between(flamengoBrasileiro, flamengoCopaDoBrasil);
        System.out.printf("%s anos, %s meses e %s dias\n", dif.getYears(), dif.getMonths(), dif.getDays() );
        // ------------------
        LocalDate proximaSegundaFeira = LocalDate.of(2017, 12, 1).with(TemporalAdjusters.nextOrSame(DayOfWeek.MONDAY));
        LocalDate ultimoDiaMes = LocalDate.of(2016, 2, 1).with(TemporalAdjusters.lastDayOfMonth());

        System.out.println(proximaSegundaFeira);
        System.out.println(ultimoDiaMes);

        Period p1 = Period.ofYears(4);
        Period p2 = p1.plusMonths(6).plusDays(15).minusMonths(3);
        System.out.println(p2);
    }

}
```

Classes Period e Duration

- Exemplo de Duration

```java
package datasnovas;

import java.time.Duration;
import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;

public class Duracao {

    public static void main(String[] args) {
        Instant hoje = Instant.parse("1985-08-17T01:45:00Z");
        Instant ontem = hoje.minus(1, ChronoUnit.DAYS);
        Instant amanha = hoje.plus(1, ChronoUnit.DAYS);

        Duration t1 = Duration.between(hoje, amanha);
        System.out.println("Diferença entre hoje e amanhã: " + t1.toDays() + " dia");
        System.out.println("Diferença entre hoje e amanhã: " + t1.toMinutes() + " minutos");

        Duration t2 = Duration.between(hoje, ontem);
        System.out.println("\nDiferença entre hoje e ontem: " + t2.toDays() + " dia");

        //-------------

        LocalDateTime ldt = LocalDateTime.parse("1985-08-17T17:45:00");
        LocalDateTime horaAnoQueVem = ldt.plusHours(1).plusYears(1);

        Duration t3 = Duration.between(ldt, horaAnoQueVem);
        System.out.println("\nDiferença entre hoje e ano que vem: " + t3.toDays() + " dias");

        //-------------

        LocalDate ld = LocalDate.parse("1985-08-17");
        LocalDate semanaQueVem = ld.plusDays(7);

        Duration t4 = Duration.between(ld.atStartOfDay(), semanaQueVem.atStartOfDay());
        System.out.println("\nDiferença entre hoje e semana que vem: " + t4.toDays() + " dias");
    }

}
```

UNOESC