Datas com Java - Parte 2

Técnicas de Programação I

Convertendo strings em datas e horas – método `parse()`

Os seguintes métodos estáticos são utilizados para criar objetos java.time de strings:

- `LocalDate.parse("YYYY-MM-DD")`
  - retorna um LocalDate com o ano *(YYYY)*, mês *(MM)* e dia *(DD)* fornecidos.
- `LocalTime.parse("hh:mm:ss")`
  - retorna um LocalTime com a hora *(hh)*, minutos *(mm)* e segundos *(ss)* fornecidos.
- `LocalDateTime.parse("YYYY-MM-DDThh:mm:ss")`
  - retorna um LocalDateTime com o ano *(YYYY)*, mês *(MM)*, dia *(DD)*, hora *(hh)*, minutos *(mm)* e segundos *(ss)* fornecidos.
  - necessário adicionar o caractere **T** entre os valores.

Convertendo strings em datas e horas – método parse()

```
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;

public class DatasCustomizadas {

    public static void main(String[] args) {
        LocalDate dataInicio = LocalDate.of(2022, 01, 01);
        LocalDate dataFim = LocalDate.parse("2022-01-30");
        System.out.println("Ferias de " + dataInicio + " ate " + dataFim);

        LocalTime horaInicio = LocalTime.of(8,15,30);
        LocalTime horaFim = LocalTime.parse("09:47:55");
        System.out.println("Intervalo de " + horaInicio + " ate " + horaFim);

        LocalDateTime dataHoraInicio = LocalDateTime.of(2022, 12,15, 1, 22, 43);
        LocalDateTime dataHoraFim = LocalDateTime.parse("2022-12-20T05:45:43");
        System.out.println("Recesso de " + dataHoraInicio + " ate " + dataHoraFim);
    }
}
```

Resultado:

```
Ferias de 2022-01-01 ate 2022-01-30
Intervalo de 08:15:30 ate 09:47:55
Recesso de 2022-12-15T01:22:43 ate 2022-12-20T05:45:43
```

Formatando datas e horas com DateTimeFormatter

Formatando datas e horas com DateTimeFormatter

A classe *DateTimeFormatter* é utilizada para armazenar formatos de data personalizados. Podem ser instanciadas através do método:

```
DateTimeFormatter.ofPattern(<pattern>)
```

O objeto *DateTimeFormatter* pode ser utilizado como parâmetro no método **parse()**, para converter strings em data/hora. Da mesma forma, pode se obter uma string de uma data/hora através do método **.format()**.

Teste o exemplo a seguir.

Datas com Java - Parte II

Técnicas de Programação I

Formatando datas e horas com DateTimeFormatter

```java
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class FormatandoDatas {

    public static void main(String[] args) {
        String strDataFormatoBR = "25-06-1980 11:15";
        DateTimeFormatter formatoData = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm");
        LocalDateTime dataHoraBR = LocalDateTime.parse(strDataFormatoBR, formatoData);
        System.out.println("Data e hora extraidos do formato BR: " + dataHoraBR);

        DateTimeFormatter outroFormato = DateTimeFormatter.ofPattern("dd MM yy - HH, mm");
        System.out.println("Outros formatos: " + dataHoraBR.format(outroFormato));
    }
}
```

Resultado:

```
Data e hora extraidos do formato BR: 1980-06-25T11:15
Outros formatos: 25 06 80 - 11, 15
```

Algumas operações com data/hora

Algumas operações com data/hora

As classes do pacote **java.time** possuem diversas operações úteis na manipulação de unidades de tempo, como subtrair e adicionar valores. Consulte a documentação de cada uma das classes para mais detalhes sobre **LocalTime**, **LocalDate** e **LocalDateTime**.

No exemplo a seguir, aplicamos os métodos **.getDayOfWeek()**, **.minusDays()** e **.plusDays()** da classe **LocalDate**.

Datas com Java - Parte II

Técnicas de Programação I

Algumas operações com data/hora

```java
import java.time.LocalDate;

public class ManipulandoDatas {

    public static void main(String[] args) {
        LocalDate dataNascimento = LocalDate.of(1980, 6, 25);
        System.out.println("Dia da Semana: " + dataNascimento.getDayOfWeek());
        System.out.println("Três dias antes: " + dataNascimento.minusDays(3));
        System.out.println("Três dias depois: " + dataNascimento.plusDays(3));
    }
}
```

Resultado:

```
Dia da Semana: WEDNESDAY
Três dias antes: 1980-06-22
Três dias depois: 1980-06-28
```

Data/hora com fuso horário

Data/hora com fuso horário

O fuso horário de **Greenwich** (*Greenwich Mean Time - GMT*) é a base das datas e horas da **java.time**. O nome do padrão que usa o **GMT** como base é denominado **Coordinated Universal Time** (*UTC*).

A classe **ZonedDateTime** representa uma data e hora contendo um fuso horário. Ela precisa de um **ZoneId** como parâmetro para identificar o fuso horário utilizado.

No exemplo a seguir, inserimos o fuso horário de **São Paulo, Acre e Lisboa** a um **LocalDateTime**, criando três objetos **ZonedDateTime** distintos.

Atente ao uso do método estático **.of()** na criação dos objetos.

Datas com Java - Parte II

Técnicas de Programação I

Data/hora com fuso horário

```java
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;

public class DataComFuso {

    public static void main(String[] args) {
        LocalDateTime dateTime = LocalDateTime.now();

        ZonedDateTime zonedDateTimeSP = ZonedDateTime.of(dateTime, ZoneId.of("America/Sao_Paulo"));
        ZonedDateTime zonedDateTimeAC = ZonedDateTime.of(dateTime, ZoneId.of("Brazil/Acre"));
        ZonedDateTime zonedDateTimePT = ZonedDateTime.of(dateTime, ZoneId.of("Europe/Lisbon"));

        System.out.println("Data hora fuso SP: " + zonedDateTimeSP);
        System.out.println("Data hora fuso AC: " + zonedDateTimeAC);
        System.out.println("Data hora fuso PT: " + zonedDateTimePT);
    }
}
```

Resultado:

```
Data hora fuso SP: 2022-05-05T00:34:41.243321900-03:00[America/Sao_Paulo]
Data hora fuso AC: 2022-05-05T00:34:41.243321900-05:00[Brazil/Acre]
Data hora fuso PT: 2022-05-05T00:34:41.243321900+01:00[Europe/Lisbon]
```

Data/hora com fuso horário

A documentação da linguagem não lista todas as zonas possíveis de serem utilizadas.

Consulte as seguintes referências:

- Documentação do DateTimeFormatter
- classe LocalTime
- classe LocalDate
- classe LocalDateTime
- Todas as Zone Id
- Java 8, DateTime API
- Java 8, Um novo tempo
- Introduction to the Java 8 Date/Time API

Horário de verão

Horário de verão – método isDaylightSavings()

Para descobrirmos se um determinado horário está sob efeito do horário de verão, a linguagem dispõe do método **isDaylightSavings()**.

```
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;

public class HorarioDeVerao {

    public static void main(String[] args) {
        LocalDateTime dateTime = LocalDateTime.parse("2018-05-01T05:00:00");
        ZoneId zoneId = ZoneId.of("America/Sao_Paulo");
        ZonedDateTime zonedDateTimeSP = ZonedDateTime.of(dateTime, zoneId);

        boolean isDaylightSaving = zoneId.getRules().isDaylightSavings(zonedDateTimeSP.toInstant());
        System.out.println("Em " + zonedDateTimeSP + " SP estava em horario de verao: " + isDaylightSaving);

        dateTime = LocalDateTime.parse("2018-12-01T05:00:00");
        zonedDateTimeSP = ZonedDateTime.of(dateTime, zoneId);
        isDaylightSaving = zoneId.getRules().isDaylightSavings(zonedDateTimeSP.toInstant());
        System.out.println("Em " + zonedDateTimeSP + " SP estava em horario de verao: " + isDaylightSaving);
    }
}

// Em 2018-05-01T05:00-03:00[America/Sao_Paulo] SP estava em horario de verao: false

// Em 2018-12-01T05:00-02:00[America/Sao_Paulo] SP estava em horario de verao: true
```

Horário de verão – método isDaylightSavings()

Do exemplo anterior, para utilizar o método **isDaylightSavings()**, foi necessário usar um objeto **ZoneRules**, retornado pelo método **getRules()** da **ZoneId** e converter a data/hora com fuso horário no tipo Instant.

Ajustes em datas e horas

Ajustes em datas e horas

Os objetos do pacote **java.time** são imutáveis, mas a API fornece a classe **TemporalAdjusters** que contempla diversos métodos que servem para ajustar uma data/hora, criando novas instâncias. Alguns métodos presentes nesta classe:

- *firstDayOfNextYear()*
- *lastDayOfMonth()*
- *withHour()*
- *withYear()*

O exemplo a seguir identifica qual foi a sexta-feira mais próxima de uma determinada data.

Ajustes em datas e horas

O exemplo a seguir identifica qual foi a sexta-feira mais próxima de uma determinada data.

```
import java.time.DayOfWeek;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.time.temporal.TemporalAdjusters;

public class AjustandoDatas {

    public static void main(String[] args) {
        LocalDateTime dateTime = LocalDateTime.parse("2018-05-01T05:00:00");
        ZonedDateTime zonedDateTimeSP = ZonedDateTime.of(dateTime, ZoneId.of("America/Sao_Paulo"));
        System.out.println("O dia " + zonedDateTimeSP + " foi no dia da semana: " + zonedDateTimeSP.getDayOfWeek());

        ZonedDateTime nextFridayDateTime = zonedDateTimeSP.with(TemporalAdjusters.next(DayOfWeek.FRIDAY));
        System.out.println("A data da proxima SEXTA foi: " + nextFridayDateTime);
    }
}
```

Resultado:

```
O dia 2018-05-01T05:00-03:00[America/Sao_Paulo] foi no dia da semana: TUESDAY
A data da proxima SEXTA foi: 2018-05-04T05:00-03:00[America/Sao_
```

Períodos, durações e instantes de tempo

Períodos, durações e instantes de tempo

Além das classes mencionadas nas seções anteriores, o pacote **java.time** também possui as classes:

- **Period** para representar um período de dias, de meses ou de anos;
- **Duration** para representar um curto período de minutos ou horas
- **Instant**, representando o número de segundos e nanossegundos desde 01/01/1970.

O exemplo a seguir mostra como calcular qual a data após 3 meses de uma data referência e mostra o mesmo horário no fuso de São Paulo, no fuso horário de Portugal.

Datas com Java - Parte II

Técnicas de Programação I

Períodos, durações e instantes de tempo

```
import java.time.Period;
import java.time.ZoneId;
import java.time.ZonedDateTime;

public class UsandoPeriodosDeTempo {

    public static void main(String[] args) {
        ZonedDateTime startDateTime = ZonedDateTime.of(2022, 4, 8, 13, 35, 56, 0, ZoneId.of("America/Sao_Paulo"));

        Period period = Period.ofMonths(3);
        ZonedDateTime endDateTime = startDateTime.plus(period);

        System.out.println("Inicio da Atividade: " + startDateTime);
        System.out.println("Fim da Atividade (+ 3 meses): " + endDateTime);

        System.out.println("\nFim da Atividade com horario de Portugal: " +
endDateTime.withZoneSameInstant(ZoneId.of("Europe/Lisbon")));
    }
}
```

Resultado:

```
Inicio da Atividade: 2022-04-08T13:35:56-03:00[America/Sao_Paulo]
Fim da Atividade (+ 3 meses): 2022-07-08T13:35:56-03:00[America/Sao_Paulo]

Fim da Atividade com horario de Portugal: 2022-07-08T17:35:56+01:00[Europe/Lisbon]
```

ada

Períodos, durações e instantes de tempo

Exemplo de cálculo de um intervalo de tempo usando o Duration:

```java
import java.time.Duration;
import java.time.LocalTime;
import java.time.temporal.ChronoUnit;

public class DuracaoDeIntervalo {

    public static void main(String[] args) {
        LocalTime begins = LocalTime.of(12, 07, 10);
        LocalTime ends = LocalTime.of(14, 22, 37);
        System.out.println("Intervalo: " + begins + " - " + ends);

        long minutes = ChronoUnit.MINUTES.between(begins, ends);
        System.out.println("Total minutos: " + begins + " - " + ends);

        Duration duration = Duration.ofMinutes(minutes);
        System.out.println("Duracao do intervalo: " + duration);
    }
}
```

Resultado:

```
Intervalo: 12:07:10 - 14:22:37
Total minutos: 12:07:10 - 14:22:37
Duracao do intervalo: PT2H15M
```

Períodos, durações e instantes de tempo

Finalmente, o exemplo a seguir ilustra o uso do Instant:

```
import java.time.Instant;
import java.time.ZoneId;
import java.time.ZonedDateTime;

public class UsandoInstant {

    public static void main(String[] args) {
        ZonedDateTime eventDateTime = ZonedDateTime.of(1998, 1, 13, 16, 45, 56, 0, ZoneId.of("America/Sao_Paulo"));
        Instant eventInstant = eventDateTime.toInstant();
        System.out.println("Data e hora do evento: " + eventDateTime);
        System.out.println("Instante do evento: " + eventInstant);

        System.out.println("\nNumero de segundos de 01-Janeiro-1970 ate o inicio do evento: " +
eventInstant.getEpochSecond());
    }
}
```

Resultado:

```
Data e hora do evento: 1998-01-13T16:45:56-02:00[America/Sao_Paulo]
Instante do evento: 1998-01-13T18:45:56Z

Numero de segundos de 01-Janeiro-1970 ate o inicio do evento: 884717156
```

Internacionalizando o formato de datas e horas

Internacionalizando o formato de datas e horas

A classe **java.util.Locale** representa uma região geográfica, política ou cultural específica. Ela pode ser usada com um **DateTimeFormatter** para personalizar uma saída formatada para um local específico.

Construtores na classe Locale:

```
Locale(String language)
Locale(String language, String country)
Locale(String language, String country, String variant)
```

Datas com Java - Parte II

Técnicas de Programação I

Internacionalizando o formato de datas e horas

Veja como formatar datas e horas no formato brasileiro, japonês e dos EUA →

Resultado:

```
Formato longo:
Brasil: 5 de maio de 2022 13:14:16 CEST
Japao: 2022年5月5日 13:14:16 CEST
USA: 2022 May 5 13:14:16 CEST

Formato curto:
Brasil: 05/05/2022 13:14
Japao: 2022/05/05 13:14
USA: 2022-05-05 13:14
```

```java
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.util.Locale;

public class UsandoLocale {

    public static void main(String[] args) {
        Locale locBR = new Locale("pt", "BR");
        Locale locJA = new Locale("ja");
        Locale locUS = new Locale("US");

        ZonedDateTime zonedDateTime = ZonedDateTime.now();

        System.out.println("Formato longo: ");
        System.out.println("Brasil: " +
                zonedDateTime.format(
                        DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG)
                                .withLocale(locBR)));
        System.out.println("Japao: " +
                zonedDateTime.format(
                        DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG)
                                .withLocale(locJA)));
        System.out.println("USA: " +
                zonedDateTime.format(
                        DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG)
                                .withLocale(locUS)));

        System.out.println("\nFormato curto: ");
        System.out.println("Brasil: " +
                zonedDateTime.format(
                        DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                                .withLocale(locBR)));
        System.out.println("Japao: " +
                zonedDateTime.format(
                        DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                                .withLocale(locJA)));
        System.out.println("USA: " +
                zonedDateTime.format(
                        DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                                .withLocale(locUS)));
    }
}
```

Resumo

Datas com Java - Parte II

Técnicas de Programação I

Resumo

Nas tabela, listamos os principais métodos **java.api** para serem usados e suas classes →

classe java.time	exemplo de método
LocalDate	*LocalDate.now();* *LocalDate.of(2022, 8, 22);* *LocalDate.parse("2022-8-22");*
LocalTime	*LocalTime.now();* *LocalTime.of(11, 20, 37);* *LocalTime.of("11:20:37");*
LocalDateTime	*LocalDateTime.now();* *LocalDateTime.of(aDate, aTime);* *LocalDateTime.parse("2022-05-09T11:20:37");* *LocalDateTime.parse(aDateTime, aFormatter);* *LocalDateTime.parse("2022-09-22T11:20", aFormatter);*
ZonedDateTime	*ZonedDateTime.now();* *ZonedDateTime.of(aDateTime, ZoneId.of(aZoneString));* *ZonedDateTime.parse("2018-05-09T11:20:47-03:00");*
OffsetDateTime	*OffsetDateTime.now();* *OffsetDateTime.of(aDateTime, ZoneOffset.of("-03:00"));* *OffsetDateTime.parse("2018-04-08T11:19:36-03:00");*
format.DateTimeFormatter	*DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");* *DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT).withLocale(aLocale);*
Instant	*Instant.now();* *zonedDateTime.toInstant();* *aDateTime.toInstant(ZoneOffset.of("+5"));*
Duration	*Duration.between(aDate1, aDate2);* *Duration.ofMinutes(5);*
Period	*Period.between(aDate1, aDate2);* *Period.ofDays(3);*
util.Locale	*Locale.getDefault();* *new Locale(String language);* *new Locale(String language, String country);*

ada

Datas com Java - Parte II

Técnicas de Programação I

Resumo

Principais métodos do pacote **java.time** →

java.time Class	exemplo de ajustes
LocalDate	*aDate.minusDays(3);* *aDate.plusWeeks(1);* *aDate.withYear(2018);*
LocalTime	*aTime.minus(3, ChronoUnit.MINUTES);* *aTime.plusMinutes(3);* *aTime.withHour(12);*
LocalDateTime	*aDateTime.minusDays(3);* *aDateTime.plusMinutes(10);* *aDateTime.plus(Duration.ofMinutes(5));* *aDateTime.withMonth(2);*
ZonedDateTime	*zonedDateTime.withZoneSameInstant(ZoneId.of("US/Pacific"));*

ada

Exercícios

Exercício 1: Controle de Férias do Funcionário

Crie um programa que:

1. Leia do usuário:
   - data de início das férias (formato yyyy-MM-dd);
   - quantidade de dias de férias.

2. Calcule:
   - data de término das férias;
   - dia da semana de início e de término;
   - data limite para entregar as demandas: 3 dias antes da data início.

3. Exiba todas as informações.

Exercício 2: Tempo de Espera em Agência

A agência funciona das 10:00 às 16:00.

Crie um programa que:

1. Leia do usuário o horário de chegada (HH:mm).

2. Diga se chegou:
   - antes de abrir,
   - durante o atendimento,
   - após o fechamento.

3. Se chegou durante o atendimento, informe quantos minutos faltam para o fechamento.

Exercício 3: Calculadora de Boleto

Crie um programa que:

1. Leia a data de vencimento de um boleto (formato dd/MM/yyyy).

2. Calcule:
   a. data do desconto (5 dias antes),
   b. início dos juros (3 dias depois).

3. Ajuste qualquer data que cair em sábado ou domingo para o próximo dia útil.

4. Exiba todas as informações formatadas.

Exercício 4: Tempo Decorrido de Atendimento

Na Caixa, o sistema registra início e fim de atendimentos.

Crie um programa que:

1. Receba do usuário:
    a. início do atendimento (formato yyyy-MM-ddTHH:mm);
    b. fim do atendimento (mesmo formato).

2. Calcule:
    a. duração em minutos,
    b. duração em horas,
    c. duração total em segundos.

3. Exiba tudo no console.

Exercício 5: Dashboard de Datas de Contratos Caixa

Crie um programa que:

1. Leia duas datas:
    a. data de início de um contrato (yyyy-MM-dd)
    b. data atual (yyyy-MM-dd)

2. Calcule há quanto tempo o contrato está ativo, exibindo:
    a. anos
    b. meses
    c. dias

3. Se o contrato tiver mais de 5 anos, exiba uma mensagem:
    a. *"Contrato apto para renovação automática"*
       Senão:
    b. "Contrato ainda não atingiu o prazo mínimo"

Obrigada