Java 8 Date

Avec Java 8, une nouvelle API date-heure est introduite pour couvrir les inconvénients suivants de l'ancienne API date-heure.

Non thread-safe - java.util.Date n'est pas thread-safe, les développeurs doivent donc faire face à un problème de concurrence lors de l'utilisation de la date. La nouvelle API date-heure est immuable et n'a pas de méthodes de définition.

Mauvaise conception - La date par défaut commence à partir de 1900, le mois commence à partir de 1 et le jour commence à partir de 0, donc pas d'uniformité. L'ancienne API avait des méthodes moins directes pour les opérations de date. La nouvelle API fournit de nombreuses méthodes utilitaires pour de telles opérations.

Gestion difficile des fuseaux horaires − Les développeurs ont dû écrire beaucoup de code pour gérer les problèmes de fuseau horaire. La nouvelle API a été développée en gardant à l'esprit la conception spécifique au domaine.

Java 8 introduit une nouvelle API date-heure sous le package java.time. Voici quelques-unes des classes importantes introduites dans le package java.time.

Local - API date-heure simplifiée sans complexité de gestion du fuseau horaire.

Zoned - API date-heure spécialisée pour gérer différents fuseaux horaires.

LocalDate/LocalTime et LocalDateTime

Les classes LocalDate/LocalTime et LocalDateTime simplifient le développement là où les fuseaux horaires ne sont pas nécessaires.

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

public class Java8Tester {

   public static void main(String args[]) {
      Java8Tester java8tester = new Java8Tester();
      java8tester.testLocalDateTime();
```

```
   }

   public void testLocalDateTime() {
      // Get the current date and time
      LocalDateTime currentTime = LocalDateTime.now();
      System.out.println("Current DateTime: " + currentTime);

      LocalDate date1 = currentTime.toLocalDate();
      System.out.println("date1: " + date1);

      Month month = currentTime.getMonth();
      int day = currentTime.getDayOfMonth();
      int seconds = currentTime.getSecond();

      System.out.println("Month: " + month +"day: " + day +"seconds: " + seconds);

      LocalDateTime date2 = currentTime.withDayOfMonth(10).withYear(2012);
      System.out.println("date2: " + date2);

      //12 december 2014
      LocalDate date3 = LocalDate.of(2014, Month.DECEMBER, 12);
      System.out.println("date3: " + date3);

      //22 hour 15 minutes
      LocalTime date4 = LocalTime.of(22, 15);
      System.out.println("date4: " + date4);

      //parse a string
      LocalTime date5 = LocalTime.parse("20:15:30");
      System.out.println("date5: " + date5);
   }
}
```

Il devrait produire la sortie suivante :

```
Current DateTime: 2014-12-09T11:00:45.457
date1: 2014-12-09
Month: DECEMBERday: 9seconds: 45
date2: 2012-12-10T11:00:45.457
date3: 2014-12-12
```

```
date4: 22:15
date5: 20:15:30
```

ZonedDateTime

L'API date-heure zonée doit être utilisée lorsque le fuseau horaire doit être pris en compte.

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

public class Java8Tester {

   public static void main(String args[]) {
      Java8Tester java8tester = new Java8Tester();
      java8tester.testZonedDateTime();
   }

   public void testZonedDateTime() {
      // Get the current date and time
      ZonedDateTime date1 = ZonedDateTime.parse("2007-12-03T10:15:30+05:30[Asia/Karachi]");
      System.out.println("date1: " + date1);

      ZoneId id = ZoneId.of("Europe/Paris");
      System.out.println("ZoneId: " + id);

      ZoneId currentZone = ZoneId.systemDefault();
      System.out.println("CurrentZone: " + currentZone);
   }
}
```

Il devrait produire la sortie suivante :

```
date1: 2007-12-03T10:15:30+05:00[Asia/Karachi]
ZoneId: Europe/Paris
CurrentZone: Etc/UTC
```

ChronoUnit

L'énumération java.time.temporal.ChronoUnit est ajoutée dans Java 8 pour remplacer les valeurs entières utilisées dans l'ancienne API pour représenter le jour, le mois, etc.

```
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;

public class Java8Tester {

   public static void main(String args[]) {
      Java8Tester java8tester = new Java8Tester();
      java8tester.testChromoUnits();
   }

   public void testChromoUnits() {
      //Get the current date
      LocalDate today = LocalDate.now();
      System.out.println("Current date: " + today);

      //add 1 week to the current date
      LocalDate nextWeek = today.plus(1, ChronoUnit.WEEKS);
      System.out.println("Next week: " + nextWeek);

      //add 1 month to the current date
      LocalDate nextMonth = today.plus(1, ChronoUnit.MONTHS);
      System.out.println("Next month: " + nextMonth);

      //add 1 year to the current date
      LocalDate nextYear = today.plus(1, ChronoUnit.YEARS);
      System.out.println("Next year: " + nextYear);

      //add 10 years to the current date
      LocalDate nextDecade = today.plus(1, ChronoUnit.DECADES);
      System.out.println("Date after ten year: " + nextDecade);
   }
}
```

La sortie est

```
Current date: 2014-12-10
Next week: 2014-12-17
Next month: 2015-01-10
Next year: 2015-12-10
Date after ten year: 2024-12-10
```

Duration

Avec Java 8, deux classes spécialisées sont introduites pour gérer les décalages horaires.

Période - Il traite de la durée basée sur la date.

Durée - Il s'agit d'une durée basée sur le temps.

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

import java.time.LocalDate;
import java.time.LocalTime;
import java.time.Duration;
import java.time.Period;

public class Java8Tester {

   public static void main(String args[]) {
      Java8Tester java8tester = new Java8Tester();
      java8tester.testPeriod();
      java8tester.testDuration();
   }

   public void testPeriod() {
      //Get the current date
      LocalDate date1 = LocalDate.now();
      System.out.println("Current date: " + date1);

      //add 1 month to the current date
      LocalDate date2 = date1.plus(1, ChronoUnit.MONTHS);
      System.out.println("Next month: " + date2);

      Period period = Period.between(date2, date1);
      System.out.println("Period: " + period);
```

```
   }

   public void testDuration() {
      LocalTime time1 = LocalTime.now();
      Duration twoHours = Duration.ofHours(2);

      LocalTime time2 = time1.plus(twoHours);
      Duration duration = Duration.between(time1, time2);

      System.out.println("Duration: " + duration);
   }
}
```

Compatibilité

Une méthode toInstant() est ajoutée aux objets Date et Calendar d'origine, qui peuvent être utilisés pour les convertir vers la nouvelle API Date-Heure. Utilisez une méthode ofInstant(Insant,ZoneId) pour obtenir un objet LocalDateTime ou ZonedDateTime.

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

import java.util.Date;

import java.time.Instant;
import java.time.ZoneId;

public class Java8Tester {

   public static void main(String args[]) {
      Java8Tester java8tester = new Java8Tester();
      java8tester.testBackwardCompatability();
   }

   public void testBackwardCompatability() {
      //Get the current date
      Date currentDate = new Date();
      System.out.println("Current date: " + currentDate);
```

```
        //Get the instant of current date in terms of milliseconds
        Instant now = currentDate.toInstant();
        ZoneId currentZone = ZoneId.systemDefault();

        LocalDateTime localDateTime = LocalDateTime.ofInstant(now, currentZone);
        System.out.println("Local date: " + localDateTime);

        ZonedDateTime zonedDateTime = ZonedDateTime.ofInstant(now, currentZone);
        System.out.println("Zoned date: " + zonedDateTime);
    }
}
```

Il devrait produire la sortie:

```
Current date: Wed Dec 10 05:44:06 UTC 2014
Local date: 2014-12-10T05:44:06.635
Zoned date: 2014-12-10T05:44:06.635Z[Etc/UTC]
```

Revision #2
Created 4 June 2022 18:10:20 by ggpilou2
Updated 19 June 2022 10:52:23 by ggpilou2