JAVA 8 - NEW DATE/TIME API

With Java 8, a new Date-Time API is introduced to cover the following drawbacks of old date-time API −

- **Not thread safe** − java.util.Date is not thread safe, thus developers have to deal with concurrency issue while using date. The new date-time API is immutable and does not have setter methods.
- **Poor design** − Default Date starts from 1900, month starts from 1, and day starts from 0, so no uniformity. The old API had less direct methods for date operations. The new API provides numerous utility methods for such operations.
- **Difficult time zone handling** − Developers had to write a lot of code to deal with timezone issues. The new API has been developed keeping domain-specific design in mind.

Java 8 introduces a new date-time API under the package **java.time**. Following are some of the important classes introduced in java.time package −

- **Local** − Simplified date-time API with no complexity of timezone handling.
- **Zoned** − Specialized date-time API to deal with various timezones.

Local Data-Time API

LocalDate/LocalTime and LocalDateTime classes simplify the development where timezones are not required. Let's see them in action −

Java8Tester.java

```
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);
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows −

```
$javac Java8Tester.java
```

Now run the Java8Tester as follows −

```
$java Java8Tester
```

It should produce the following output −

```
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
```

Zoned Data-Time API

Zoned date-time API is to be used when time zone is to be considered. Let us see them in action −

Java8Tester.java

```
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);
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows −

```
$javac Java8Tester.java
```

Now run the Java8Tester as follows −

```
$java Java8Tester
```

It should produce the following output −

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

Chrono Units Enum

java.time.temporal.ChronoUnit enum is added in Java 8 to replace the integer values used in old API to represent day, month, etc. Let us see them in action −

Java8Tester.java

```
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);
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows −

```
$javac Java8Tester.java
```

Now run the Java8Tester as follows −

```
$java Java8Tester
```

It should produce the following result −

```
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
```

Period & Duration

With Java 8, two specialized classes are introduced to deal with the time differences −

- **Period** − It deals with date based amount of time.
- **Duration** − It deals with time based amount of time.

Let us understand them with an example −

Java8Tester.java

```
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);
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows −

```
$javac Java8Tester.java
```

Now run the Java8Tester as follows −

```
$java Java8Tester
```

It should produce the following output −

```
Current date: 2014-12-10
Next month: 2015-01-10
Period: P-1M
Duration: PT2H
```

Temporal Adjusters

TemporalAdjuster is used to perform the date mathematics. For example, get the "Second Saturday of the Month" or "Next Tuesday". Let us see an example of it −

Java8Tester.java

```
import java.time.LocalDate;
import java.time.temporal.TemporalAdjusters;
import java.time.DayOfWeek;

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

   public void testAdjusters(){

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

      //get the next tuesday
      LocalDate nextTuesday = date1.with(TemporalAdjusters.next(DayOfWeek.TUESDAY));
      System.out.println("Next Tuesday on : " + nextTuesday);

      //get the second saturday of next month
      LocalDate firstInYear = LocalDate.of(date1.getYear(),date1.getMonth(), 1);
      LocalDate secondSaturday =
firstInYear.with(TemporalAdjusters.nextOrSame(DayOfWeek.SATURDAY)).with(TemporalAdjusters.
next(DayOfWeek.SATURDAY));
      System.out.println("Second Saturday on : " + secondSaturday);
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows −

```
$javac Java8Tester.java
```

Now run the Java8Tester as follows −

```
$java Java8Tester
```

It should produce the following result −

```
Current date: 2014-12-10
Next Tuesday on : 2014-12-16
Second Saturday on : 2014-12-13
```

Backward Compatibility

A **toInstant** method is added to the original Date and Calendar objects, which can be used to convert them to the new Date-Time API. Use an ofInstant*Insant*, *ZoneId* method to get a LocalDateTime or ZonedDateTime object. Let us understand it with an example −

Java8Tester.java

```
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);
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows −

```
$javac Java8Tester.java
```

Now run the Java8Tester as follows −

```
$java Java8Tester
```

It should produce the following output −

```
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]
```