Java 8 Hands on Lab

Magdalena Petrova, Silviya Brayanova, Ivan St. Ivanov

Version 0.1
May 1, 2015

Table of Contents

Chapter 1. Introduction

The Java Language and Platform have matured a lot in the last 20 years. Born as a blue collar language, which can be understood and used by the average developer, Java stayed like that throughout all these years to become top choice for contemporary developers and projects. With every new release the language and platform engineers managed to keep the delicate balance between bringing in new features and capabilities, while maintaining the backward compatibility and the simplicity.

Looking at Java's 20-year history, one can identify two big groups of releases. The Java architects Brian Goetz and Mark Reinhold have identified those groups as evolutionary and revolutionary. Java 5 might seem as the first revolutionary release – it brought things like generics, annotations, enums, static imports, autoboxing and unboxing of primitive types, the enhanced foreach loop and varargs. But don't forget J2SE 1.2 before it. It introduced the Just In Time (JIT) compilation of the interpreted bytecode and the collection framework among others.

After two evolutionary versions (Java 6 and 7), Java SE 8 came out with many long anticipated changes. As Oracle already announced this April the end of public updates of Java SE 7, it is high time the Java developers become aware of what is new in version 8 and how the new features should be applied to the existing and upcoming projects.

This Hands on Lab will guide you in finding which those features are and why and how you should use them.

Chapter 2. Lambda functions

Chapter 3. The stream API

Chapter 4. The Date and Time API

The new Date Time API in java 8 include new classes and utility method which made working with date and time easier than before.

The main API for dates, times, instants, periods and duration are included in package `java.time`

Previous to Java 8, to calculate the time one day in the future you would need to write something like the following:

```
 Date currentDate = new Date (System.currentTimeMillis());
 Date nextDay = new Date( currentDate.getTime() + 1 * 24 * 60 * 60 * 1000); // add 24
hours
```

In Java 8, you can more simply write the following:

```
1 LocalTime today = LocalTime.now();
2 LocalTime nextDay = now.plus(24, HOURS);
```

You also can use the followign well-named methods such as plusDays, plusMonths, minusDays, and minusMonths. For example:

```
1 LocalDate today = LocalDate.now();
2 LocalDate nextDay = today.plusDays(1);

3 LocalDate nextMonth = today.plusMonths(1);
4 LocalDate aMonthAgo = today.minusMonths(1);
```

IMPORTANT: ach method returns a different instance of LocalDate.This is because the new Date-Time types are immutable.

4.1. The new Date Time classes ：

- `LocalDate` – Represents a date without timezone, often viewed as year-month-day.
- `LocalTime` – Time of day time without time-zone.
- `LocalDateTime` – Includes date and time without time-zone.
- `ZonedDateTime` - It represents date-time with a time-zone in the ,such as 2007-12-03T10:15:30+01:00 Europe/Paris.
- `OffsetDateTime` - Date and time with a corresponding time zone offset from Greenwich/UTC,

without a time zone ID.

- `OffsetTime` - Time with time zone offset from Greenwich/UTC, without a time zone ID.

4.2. Classes for Timezone:

- `ZoneId` - A time-zone ID, such as Europe/Paris
- `ZoneOffset` - this is time zone offset from Greenwich/UTC time

4.3. Class : Instant

It represents time in `nanoseconds`. The method `toInstant` can be used in `java.util.Date` for converting of the Date from the old to the new date time API.

For example:

```
Date date = new Date();
Instant instant= date.toInstant();
LocalDateTime dateTime = ZonedDateTime.ofInstant(instant, timeZone );
```

4.4. Enums:

Package : `java.time.temporal.ChronoUnit` includes enums for **"days"**, **"hours"** , **"months"**, like:

- `ChronoUnit.WEEKS`
- `ChronoUnit.MONTHS`
- `ChronoUnit.YEARS`
- `ChronoUnit.DECADES`

There's also the `java.time.DayOfWeek` and `java.time.Month` enums. **For example: DayOfWeek.MONDAY**

4.5. Class Clock

Clock can be used instead of `System.currentTimeMillis()` .

Class Clock and its methods ***fixed*** or ***offset*** can be used in test scenarios where the time is need to be changed.

4.6. Classes for period and duration

- `Period` - It represents periods , like: **2 years, 3 months and 4 days**

- `Duration` - This is amount of time in terms of ***hours, seconds, minutes, nanoseconds*** . **For example: “54.5 minutes”**

4.7. Class java.time.temporal.TemporalAdjusters

This class contains a standard set of adjusters, available as static methods. These include:

- Finding the **first** or **last day of the month** - *firstDayOfMonth()*, *lastDayOfMont()*
- Finding the **first day of next month** - *firstDayOfNextMonth()*
- Finding the **first or last day of the year** - *firstDayOfYear()*
- Finding the **first day of next year** - *firstDayOfNextYear()*
- Finding the **first or last day-of-week within a month** , such as **"first Wednesday in June"**
- Finding the **next or previous day-of-week**, such as **"next Thursday"** - *next(DayOfWeek)*, *nextOrSame(DayOfWeek)*, *previous(DayOfWeek)*, *previousOrSame(DayOfWeek)*

4.8. Class: DateTimeFormatter

This class formats the date for printing and parsing.

4.9. Exercise

From 01.01.2017 from 16.00 (Bulgarian time) start film festival. Every day there is projection in Sofia and Berlin of one movie from the list, like the first movie projection is on 01.01.2017, the second movie will be projected on 02.01.2017 and etc.

Print of the screen the date and start time and end time of the movie projection in each city. Please note that every movie has property “duration”.

4.9.1. Example:

Movie [title=Circus, The, year=1928, duration=135] date: 01.01.2017 : Bulgaria: start time:18:00 end time:20:15 , Berlin: start time: 17:00 end time: 19:15 Movie [title=Animal Crackers, year=1930, duration=123] date: 02.01.2017 : Bulgaria: start time:18:00 end time:20:03 , Berlin: start time: 17:00 end time: 19:03 ...

4.9.2. Solution:

```
private static void printMovieProjectionData(List<Movie>movies){
      ZoneId zoneIDBerlin = ZoneId.of("Europe/Berlin");
      LocalDateTime  startDateTime = LocalDateTime.of(2017, 1, 1, 17, 0) ;
      DateTimeFormatter formater = DateTimeFormatter.ofPattern("dd-M-yyyy hh:mm:ss a");

      for (Movie movie : movies) {
          int duration = movie.getDuration();
          LocalDateTime  endDateTime = startDateTime.plusMinutes(duration);

          System.out.println( movie);
          System.out.println("Sofia Start Date and Time :"+ startDateTime.format(
formater)  + " End Date and Time : "+ endDateTime.format(formater));
          System.out.println("Berlin Start Date and Time :"+ ZonedDateTime.of
(startDateTime, zoneIDBerlin).format(DateTimeFormatter.ISO_INSTANT)+
                          " Date and End Time : "+  ZonedDateTime.of(endDateTime,
zoneIDBerlin).format(DateTimeFormatter.ISO_INSTANT));

          startDateTime = startDateTime.plusDays(1);

          System.out.println();

        }
    }
```

Chapter 5. The Optional class

Chapter 6. Concurrency Improvements

Chapter 7. Default and Static Methods in Interfaces

Chapter 8. New JDK APIs