Day10: Handling Date and Time in Java8 and Collection framework Introduction

Handling Date and Time in Java8:

Until Java 1.7 version the classes present in **Java.util** package to handle Date and Time (like **Date,**
Calendar, TimeZone, etc.) are not up to the mark with respect to convenience and performance.

To overcome this problem in the 1.8 version oracle people introduced a new date and time API belonging to **java.time** package.

This java 8 date API provides various classes and Enums in **java.time** package, using which we can represent Date and Time in our application efficiently.

Some of the Important classes of the java.time package is:

1. **java.time.LocalDate:** It represents a year-month-day in the ISO calendar and is useful for representing a date without a time. It can be used to represent a date only information such as birth date or joining date.
2. **java.time.LocalTime:** It deals in time only. It is useful for representing the time of day, such as movie times, or the opening and closing times of the local library.
3. **java.time.LocalDateTime:** It handles both date and time, without a time zone. It is a combination of LocalDate with LocalTime class.

4. **java.time.Duration:** It is used to define the difference between times in time-based values (hour, minute, second, nanoseconds).
5. **java.time.Period:** It is used to define the difference between dates in date-based values (years, months, days)..
6. **java.time.format.DateTimeFormatter:** It comes up with various predefined formatter, or we can define our own. It has a parse or format method for parsing and formatting the date-time values.

To get the current Date or current Time or current date-time we need to call the static method now() on the either LocalDate, LocalTime or LocalDateTime class:

Example:

```
class Main{
    public static void main(String args[]){

        // the current date (yyyy-MM-dd) format
        LocalDate date = LocalDate.now();
        System.out.println("the current date is :"+ date);


        // the current time hh:mm:ss.nanosec
        LocalTime time = LocalTime.now();
        System.out.println("the current time is "+ time);

        // will give us the current time and date combined with "T"
        LocalDateTime current = LocalDateTime.now();
        System.out.println("current date and time : "+ current);
    }
}
```

LocalDate class:

The *LocalDate class* represents **a date in ISO format (yyyy-MM-dd) without time.** We can use it to store dates like dateOfBirth, joiningDate, etc.

It is an Immutable class, so if we try to perform any update in this object, it will return a new LocalDate class object.

There are various static and non-static methods are available to this class, by using we can handle data in java.

Example: to get the current Date in yyyy-MM-dd format:

```
LocalDate localDate = LocalDate.now();
```

We can get the *LocalDate* representing a specific day, month and year by using the static

method ***of.***

Example:

```
LocalDate dob = LocalDate.of(1985, 02, 20);//(yyyy-MM-dd)
```

Example: now(), minusDay() plusDay() methods of LocalDate class:

```
import java.time.LocalDate;
public class Main {

  public static void main(String[] args) {

    LocalDate date = LocalDate.now();

    LocalDate yesterday = date.minusDays(1);
    LocalDate tomorrow = yesterday.plusDays(2);

    System.out.println("Today date: "+date);
    System.out.println("Yesterday date: "+yesterday);
    System.out.println("Tomorrow date: "+tomorrow);
  }
}
```

Example: demonstrate isLeapYear() method of LocalDate Class:

```
import java.time.LocalDate;
public class Main{

   public static void main(String[] args) {

    LocalDate date1 = LocalDate.of(2017, 1, 13);
    System.out.println(date1.isLeapYear());

    LocalDate date2 = LocalDate.of(2016, 9, 23);
    System.out.println(date2.isLeapYear());
  }
}

Output:
false
true
```

Example: atTime() method:

```
import java.time.*;
public class Main {
  public static void main(String[] args) {

    LocalDate date = LocalDate.of(2017, 1, 13);
    LocalDateTime datetime = date.atTime(1,50,9);

    System.out.println(datetime);
  }
}

output:
2017-01-13T01:50:09
```

Formatting a date-time in user defined format:

To format a LocalDate, LocalTime, LocalDateTime object in user-defined String object we need to use **java.time.format.DateTimeFormatter** class object.

This **DateTimeFormatter** class has a static method called :

```
public static DateTimeFormatter ofPattern(String pattern);
```

To call the above method we need to pass a proper pattern in the form of String object, to which pattern we want to format the LocalDate, LocalTime, or LocalDateTime object.

We need to create a pattern with the help of following symbols:

G - Era(AD BC)

y - year(yy(18) or yyyy(2018))

M - Month(M(9) or MM(09) or MMM(Sep))(MMMMM--September)

d - day(d(23) or dd(23) or ddd(023))

E - day in a weak(E (sun))(EEEE--Sunday)

a - am pm

h - hour in am or pm (1-12)

hh - hour in am or pm (01-12)

H - hour of day in 24 hour form (0-23)

HH - hour of day in 24 hour form (00-23)

m - minute (4)

mm - minute (04)

s - second (4)

ss - second(04)

Example:

```
DateTimeFormatter formatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
```

In order to format a LocalDate or LocalTime or LocalDateTime object we need to call "format(-)" method on that object by supplying DateTimeFormatter obj.

Example: Formatting Date and Time:

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

class Main{
    public static void main(String args[]){

        DateTimeFormatter formatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
        LocalDateTime current = LocalDateTime.now();

        System.out.println("current date and time in default format: "+ current);

        String udf= current.format(formatObj);

        System.out.println("current date and time in userdefined format: "+ udf);

    }
}
```

Example: Formatting only Date:

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

class Main {
    public static void main(String args[]) {

        LocalDate date = LocalDate.now();

        System.out.println("current date in default format: " + date);

        DateTimeFormatter formatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy");

        String udf = date.format(formatObj);

        System.out.println("current date in userdefined format: " + udf);
```

```
    }
}
```

Parsing a appropriate String object to the LocalDate, LocalTime, LocalDateTime object:

In order to parse an appropriate string to the these objects we need to call parse() method on these objects.

parse() is a static method defined inside all the 3 classes (LocalDate, LocalTime, LocalDateTime), parse() method takes 2 arguments: 1. String object, 2 DateTimeFormatter object.

Example: parsing a String Date to the LocalDate object

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

class Main {
    public static void main(String args[]) {

        String dob="05/02/1985";


        DateTimeFormatter dtf=DateTimeFormatter.ofPattern("dd/MM/yyyy");

        LocalDate ld=LocalDate.parse(dob,dtf);

        System.out.println(ld);

    }
}
```

I Problem:

Let’s create an application, in which ask the user to enter his date of birth in the form of string

(dd-MM-yyyy), and print the name of Day of his date of birth.

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;

class Main {
    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Date of birth in dd-MM-yyyy pattern");
        String dob= sc.next();

        DateTimeFormatter dtf=DateTimeFormatter.ofPattern("dd-MM-yyyy");

        try {
            LocalDate date = LocalDate.parse(dob, dtf);

            String result= date.format(DateTimeFormatter.ofPattern("EEEE"));

            System.out.println("Your Birthday day is :"+result);

        }catch (Exception e){
            System.out.println("please pass the date in proper format");
        }
    }
}
```

Period and Duration class:

java.time.Period: It is used to define the difference between dates in date-based values (years, months, days).

java.time.Duration: It is used to define the difference between times in time-based values (hour, minute, second, nanoseconds).

Period Class:

This class gives amount of time between two date in the term of days, months, years.

Example: getting difference between 2 LocalDate object:

```
import java.time.LocalDate;
import java.time.Period;

class Main {
    public static void main(String args[]) {

        LocalDate cdate = LocalDate.now();

        LocalDate dob = LocalDate.of(1985, 02, 05);

        Period diff = Period.between(cdate, dob);

        System.out.println(diff);
    }
}

output:
P-37Y-1M-10D
```

Other methods of Period class:

1. public Period between(LocalDate ld1,LocalDate ld2);

   return difference in the form of Period object example: above application
2. public int getDays() //10
3. public int getMonths() //1
4. public int get Years() //37
5. public long toTotalMonths(); // it return the total number of month in that Period.

We Problem:

Let’s create an application, in which user will enter his date of birth in the form of String in

“dd-MM-yyyy” format**.** and verify whether he is eligible to cast his vote or not. (age > 18)

```
import java.time.LocalDate;
import java.time.Period;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;

class Main {
    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter your date of birth in dd-MM-yyyy format");
        String dob = sc.next();

        try{

            LocalDate d1 = LocalDate.parse(dob, DateTimeFormatter.ofPattern("dd-MM-yyyy"));

            LocalDate c1 = LocalDate.now();

            Period p = Period.between(d1,c1);

            if(p.getYears() >= 18)
                System.out.println("You are Eligible to Vote");
            else
                System.out.println("Sorry you are not eligible to Vote");


        }catch (Exception e){
            System.out.println("Please enter Date of bith in valid pattern");
        }

    }
}
```

ChronoUnit:

This ChronoUnit enum belongs to **java.time.temporal** package. it is the most useful structure in java.time API.

ChronoUnit is an **enum** that implements the ***TemporalUnit*** interface, which provides the standard units used in the **Java Date Time API.**

With the help of this enum also we can get the difference between 2 dates in more precise way.

Example:

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

class Main {
    public static void main(String args[]) {

        LocalDateTime oldDate = LocalDateTime.of(1982, 5, 31, 10, 20, 55);

        LocalDateTime newDate = LocalDateTime.of(2016, 9, 9, 10, 21, 56);

        System.out.println(oldDate);
        System.out.println(newDate);


        System.out.println(ChronoUnit.YEARS.between(oldDate, newDate) + " years");
        System.out.println(ChronoUnit.MONTHS.between(oldDate, newDate) + " months");
        System.out.println(ChronoUnit.WEEKS.between(oldDate, newDate) + " weeks");
        System.out.println(ChronoUnit.DAYS.between(oldDate, newDate)+ " days");
        System.out.println(ChronoUnit.HOURS.between(oldDate, newDate) + " hours");
        System.out.println(ChronoUnit.MINUTES.between(oldDate, newDate) + " minutes");
        System.out.println(ChronoUnit.SECONDS.between(oldDate, newDate) + " seconds");
        System.out.println(ChronoUnit.MILLIS.between(oldDate, newDate) + " milis");
        System.out.println(ChronoUnit.NANOS.between(oldDate, newDate) + " nano");

   }
}
```

Example: Adding 3 month to the current date:

If we want to perform plus or minus operation on the LocalDate, LocalTime, LocalDateTime object, then also we can use this ChronoUnit enum.

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

class Main {
    public static void main(String args[]) {


        LocalDate ld=LocalDate.now();

        LocalDate dd= ld.plus(3, ChronoUnit.MONTHS);

        System.out.println(dd);

   }
}
```

You Problem:

- Write an application, in which user will be prompted to enter startDate and endDate in the form of String in “dd-MM-yyyy” pattern.
- If the startDate is greater than end date, show the message to the user as :

  “Start date should be smaller than End date”
- if user passes the wrong date pattern then show the message to the user as:

  “Please enter date in dd-MM-yyyy” format”
- otherwise calculate the number of days and between start date and end date.
- calculate and print the total wages (wages for 1 day is 1200).

Collection Framework Introduction:

Any group of individual objects which are represented as a single unit is known as the collection of the objects.

Collection framework: it defines several predefined classes and interfaces which can be used to represent group of object as single object.

In Java The Collection framework standardize the way in which group of objects are handled uniformly.

They are like a container, where we can store and manipulate group of object in a standard way.

Note: With the help of array also we can group of multiple object as a single entity(object).

Example:

```
Student[] students = new Student[3];
```

In the above Student array we can handle 3 student object also.

Difference between array and collection framework:

1. Arrays are having fixed size in nature. In case of arrays, we are able to add the elements up to the specified size only, we are unable to add the elements over its size, if we are trying to add elements over its size then JVM will rise an exception like "ArrayIndexOutOfBoundsException". whereas Collections are having dynamically growable nature, even if we add the elements over its size then JVM will not rise any exception.
2. In Java, by default, Arrays are able to allow homogeneous elements, if we are trying to add the elements which are not same Array data type then Compiler will rise an error like "Incompatible Types". whereas In Java, by default, Collections are

able to allow heterogeneous elements, even we add different types of elements Compiler will not rise any error.

3. In Java array does not have predefined methods to perform searching and sorting operations over the elements, in case of arrays to perform searching and sorting operations developers have to provide their own logic. whereas In case of Collections, predefined methods or predefined Collections are defined to perform Searching and sorting operations over the elements.

To represent Collection objects in java applications, JAVA has provided predefined classes and interfaces in the form of **java.util** package called as **"Collection Framework"**.

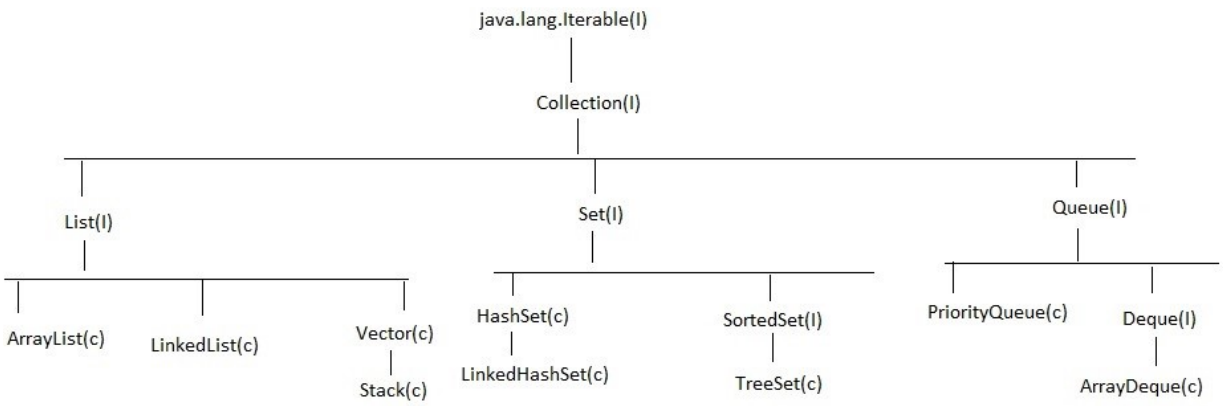
Note: In java we can group multiple object as a single object by using either by Collection or by using Map also.

Collection framework contains two section:-

1.Normal Collection (multiple objects are handled individually)
2.Map (multiple objects are handled in the form of pair (key-value)).

First we will discuss about 1st part Normal Collection, later we will see the 2nd part Map.

Collection framework major classes and interface hierarchy (Normal Collection)



Note: from the java 1.5 version onwards LinkedList class also implements the Deque interface.

The Iterable Interface:

This is the root interface for the entire collection framework. The collection interface extends the Iterable interface. Therefore, inherently, all the interfaces and classes implement this interface. The main functionality of this interface is provides the facility of iterating the elements in a forward direction only. This interface contains only one abstract method which is the iterator

```
public Iterator<T> iterator();

It returns the iterator over the elements of type T.
```

Once we get the Iterator object from any collection object, we can iterate each element one by one in a standard manner.

Methods inside the Iterator interface:

There are only three methods in the Iterator interface. They are:

1. public boolean hasNext(): It returns true if the iterator has more elements otherwise it returns false.
2. public Object next(): It returns the element and moves the cursor pointer to the next element.
3. public void remove(): It removes the elements from the Iterator.