**Date validation in java**

[**JAVA DATE**](http://beginnersbook.com/category/technology/java-guide/java-date/)

This purpose of this post is to provide step-by-step guidance to develop a utility that will have the following functionality:

1) This utility will help to **validate a date** format entered by the user.  
2) If the user entered data is valid then it will be convertible to a format that can be easily inserted into the database.

Since date can be entered by the **user in any format**, it becomes **difficult to validate** it and the format might not be acceptable to be inserted into the database as well. Thus, we need to validate whether the user entered data is in a valid format or not and then we need to convert that into a format that can be easily inserted into the database.

**Developing the utility:**

Need to import the following packages first:  
import java.sql.Date;  
import java.text.ParsePosition;  
import java.text.SimpleDateFormat;  
import java.util.ArrayList;

The following java class will provide the required functionality:

**Called program:**

[code language=”java”]  
public class DateFormatter {

public String setDisplayEffectiveDate(String date) {  
ArrayList validDateList = new ArrayList();  
validDateList.add("dd-MMM-yyyy");  
validDateList.add("dd.MM.yyyy");  
validDateList.add("dd/MM/yyyy");  
validDateList.add("MM/dd/yyyy");  
Date effectiveDate = null;  
System.out.println("setDisplayEffectiveDate1 :: date = " + date);  
if (date != null) {  
boolean correctFormat = false;  
int i = 0;  
String dateFormat = null;  
// Parse the previous string back into a Date.  
ParsePosition pos = new ParsePosition(0);  
System.out.println ("setDisplayEffectiveDate2 :: date = " + date);  
while (!correctFormat)  
{  
if (i > validDateList.size() – 1)  
{  
break;  
}  
//for (int j=0;j<validDateList.size();j++)  
//{  
dateFormat = validDateList.get(i++).toString().trim();  
System.out.println("setDisplayEffectiveDate3 :: dateFormat = "+ dateFormat);  
try  
{  
SimpleDateFormat fmt =  
new SimpleDateFormat(dateFormat);  
fmt.setLenient(false);  
effectiveDate = new Date(fmt.parse(date,pos).getTime());  
//log.debug("setDisplayEffectiveDate2 :: dateFormat = "+ dateFormat);  
//log.debug("setDisplayEffectiveDate2 :: effectiveDate = "+ effectiveDate);  
correctFormat = true;  
}  
catch (Exception e)  
{  
//log.debug("setDisplayEffectiveDate3 :: dateFormat = "+ dateFormat);  
correctFormat = false;  
}  
//}  
}  
System.out.println("setDisplayEffectiveDate :: correctFormat = " + correctFormat);  
System.out.println("setDisplayEffectiveDate2 :: effectiveDate = "+ effectiveDate);  
if (!correctFormat) {  
//effectiveDate = new Date();  
System.out.println("setDisplayEffectiveDate :: effectiveDate (error) = "+ effectiveDate);  
}

}

java.sql.Date sDate = new java.sql.Date(effectiveDate.getTime());  
SimpleDateFormat fmt1 = new SimpleDateFormat("dd-MMM-yyyy");  
return fmt1.format(sDate);  
}

}  
[/code]

**Description**

* We define a java class – ‘DateFormatter.java’ that will hold the logic to validate a date entered by the user.
* Inside the class we define the method – ‘setDisplayEffectiveDate(String date)’ with a parameter that will take the date entered by the user as a string.
* We define an array list ‘validDateList’ that will hold all the possible date formats with which the user entered date will be validated against with. For e.g.  In the above case, we are taking 4 formats only i.e. dd-MMM-yyyy, dd.MM.yyyy, dd/MM/yyyy, MM/dd/yyyy. The user entered date should belong to any one of these formats. However, if we need that more date formats are also allowed then we can keep adding them to the array list and the user entered date will be validated against that.
* Then we check if the user entered date is not null or not.

[code language=”java”]  
if (date != null) {  
….  
boolean correctFormat = false; // Setting the boolean value false initially  
ParsePosition pos = new ParsePosition(0);  
….  
}[/code]

The ParsePosition is a simple class used by Format and its subclasses to keep track of the current position during parsing. The parseObject method in the various Format classes requires a ParsePosition object as an argument.

* We go for a while loop which loops in until the user entered date matches any of the date format defined in the array list. Once it matches it quits the loop.

Inside the while loop we do the following:

[code language=”java”]  
while (!correctFormat) {  
….  
dateFormat = validDateList.get(i++).toString().trim(); // Gets the first date format from the array list  
….  
try {  
// The date format obtained above is set into the constructor of Simple Date Format class  
SimpleDateFormat fmt = new SimpleDateFormat(dateFormat);  
fmt.setLenient(false);  
// Find the effective date/time using the constructor of the Date class  
effectiveDate = new Date(fmt.parse(date,pos).getTime());  
// Setting the boolean value as ‘true’ if the effective date is obtained and quit the while loop.  
correctFormat = true;  
} catch (Exception e) {  
correctFormat = false;  
// Else set the boolean value as false and continue with the while loop.  
}  
}[/code]

**Note:**

* SimpleDateFormat is a concrete class for formatting and parsing dates in a locale-sensitive manner. It allows for formatting (date -> text), parsing (text -> date), and normalization. SimpleDateFormat allows you to start by choosing any user-defined patterns for date-time formatting.
* SimpleDateFormat.setLenient() methodsets a value that determines whether parsing is set to lenient for this DateFormat object. Leniency is determined by the ‘Calendar’ object associated with this instance. Calendar class contains methods and members to perform basic operations involving days, weeks, months, and years.
* The class ‘Date’ represents a specific instant in time, with millisecond precision.
* SimpleDateFormat.parse(String source, ParsePosition pos) method will parse a date/time string according to the given parse position.
* getTime() method provides you the total count of millisecond that have passed since 1st January 1970
* Once the Effective Date is obtained then it is formatted into a date format that can be easily inserted into the database.

[code language=”java”]  
java.sql.Date sDate = new java.sql.Date(effectiveDate.getTime());  
SimpleDateFormat fmt1 = new SimpleDateFormat("dd-MMM-yyyy");  
[/code]

* In case the user-entered date does not match any of the formats defined then it will throw an exception which will be handled in the catch block.

**Calling program:**

[code language=”java”]  
public class Start  
{  
public static void main(String[] args) {  
try{  
// Part-2 for Date Formatter  
Start start = new Start();  
DateFormatter dateformat = new DateFormatter(); // The class defined above  
String data = "26/11/1952"; // User – entered date. Given as an example here.  
// Calling the method of the Date Formatter class  
data = start.isNull(dateformat.setDisplayEffectiveDate(data));  
System.out.println("Valid Date data after formatting – "+data);  
}catch(Exception e)  
{  
System.out.println("Exception in main method-Start() :"+e);  
System.out.println("Please enter a valid date :");  
}  
}  
private String isNull(String string)  
{  
// Converts null value to a blank string  
if (string==null || "".equals(string.trim()) )  
{  
return " ";  
}  
return string.trim();  
}  
}[/code]

**Output:**

setDisplayEffectiveDate1 :: date = 26/11/1952  
setDisplayEffectiveDate2 :: date = 26/11/1952  
setDisplayEffectiveDate3 :: dateFormat = dd-MMM-yyyy  
setDisplayEffectiveDate3 :: dateFormat = dd.MM.yyyy  
setDisplayEffectiveDate3 :: dateFormat = dd/MM/yyyy  
setDisplayEffectiveDate :: correctFormat = true  
setDisplayEffectiveDate2 :: effectiveDate = 1952-11-26  
Valid Date data after formatting – 26-Nov-1952