

# Übungen zur Vorlesung Softwaretechnologie

-Wintersemester 2012/2013-

Dr. Günter Kniesel

## Übungsblatt 12 - Lösungshilfe

### Aufgabe 1. *Blackbox Test* (12 Punkte)

Gegeben sei folgende Schnittstelle:

```
interface DateParser {  
    java.util.Date parseDate(String input);  
}
```

und folgende Dokumentation:

Erzeugt ein Date-Objekt aus einem Datumsstring der Form *Tag-Monat-Jahr*.

- *Tag* und *Monat* können ein- oder zweistellig sein, evtl. mit führender 0.
- *Jahr* kann zwei- oder vierstellig sein, zweistellige Angaben beziehen sich auf das 21.-te Jahrhundert.
- Wenn *Monat* kleiner 1 ist, wird Januar angenommen, bei Werten über 12 Dezember.
- Wenn *Tag* kleiner 1 ist wird der Monatserste angenommen, bei Werten größer dem zuletzt gültigen Tag der Monatsletzte.

**Parameter:**

Der Datumsstring, der interpretiert werden soll

**Liefert zurück:**

Das interpretierte Datum (mit dem Uhrzeitwert 12:00:00) oder *null* bei ungültiger Eingabe

a) Überlegen Sie, welche Äquivalenzklassen auftreten können. Geben Sie für fünf typische korrekte Eingaben und Fehlerarten einen kritischen Eingabestring und das korrekte Methoden-Ergebnis an.

- Korrekte vollständige Eingabe
  - ◆ Testet ob richtig geparst wird
  - ◆ Voraussetzung: Gültiges Datum der Form dd-mm-yyyy
  - ◆ Beispiel: "24-10-2010" → 24.10.2010, 12:00:00
- Korrekte vollständige Eingabe mit führenden Nullen
  - ◆ Testet ob richtig geparst wird
  - ◆ Voraussetzung: Gültiges Datum der Form 0d-0m-yyyy
  - ◆ Beispiel: "04-01-2010" → 4.1.2010, 12:00:00
- Korrekte vollständige Eingabe mit verkürztem Tag/Monat
  - ◆ Testet ob richtig geparst wird
  - ◆ Voraussetzung: Gültiges Datum der Form d-m-yyyy
  - ◆ Beispiel: "4-1-2010" → 4.1.2010, 12:00:00
- Korrekte vollständige Eingabe mit zweistelliger Jahreszahl

- ◆ Testet ob richtig geparkt wird
- ◆ Voraussetzung: Gültiges Datum der Form dd-mm-yy
- ◆ Beispiel: "24-01-10" → 24.1.2010, 12:00:00
- Jahr 0 (Extremwert)
  - ◆ Testet ob richtig geparkt wird
  - ◆ Voraussetzung: Gültiges Datum der Form dd-mm-0000
  - ◆ Beispiel: "24-01-0000" → 24.1.0, 12:00:00
- Jahr 9999 (Extremwert)
  - ◆ Testet ob richtig geparkt wird
  - ◆ Voraussetzung: Gültiges Datum der Form dd-mm-9999
  - ◆ Beispiel: "24-01-9999" → 24.1.9999, 12:00:00
- Monat < 1
  - ◆ Testet ob richtig angepasst wird
  - ◆ Voraussetzung: Datum der Form dd-mm-yyyy mit mm < 0
  - ◆ Beispiel: "24-00-2010" → 24.1.2010, 12:00:00
- Monat > 12
  - ◆ Testet ob richtig angepasst wird
  - ◆ Voraussetzung: Datum der Form dd-mm-yyyy mit mm > 12
  - ◆ Beispiel: "24-13-2010" → 24.12.2010, 12:00:00
- Tag < 1
  - ◆ Testet ob richtig angepasst wird
  - ◆ Voraussetzung: Datum der Form dd-mm-yyyy mit dd < 0
  - ◆ Beispiel: "00-01-2010" → 1.1.2010, 12:00:00
- Tag > 31 (z.B Januar, März etc.)
  - ◆ Testet ob richtig angepasst wird
  - ◆ Voraussetzung: Datum der Form dd-mm-yyyy mit dd > 31 und mm = 1
  - ◆ Beispiel: "32-01-2010" → 31.1.2010, 12:00:00
- Tag > 30 (z.B April, Juni etc.)
  - ◆ Testet ob richtig angepasst wird
  - ◆ Voraussetzung: Datum der Form dd-mm-yyyy mit dd > 30 und mm = 4
  - ◆ Beispiel: "31-04-2010" → 30.4.2010, 12:00:00
- Tag > 28 (Februar, nicht Schaltjahr)
  - ◆ Testet ob richtig angepasst wird
  - ◆ Voraussetzung: Datum der Form dd-mm-yyyy mit dd > 28 und mm = 2 und yyyy ist kein Schaltjahr
  - ◆ Beispiel: "29-02-2010" → 29.2.2010, 12:00:00
- Tag > 29 (Februar, Schaltjahr)
  - ◆ Testet ob richtig angepasst wird
  - ◆ Voraussetzung: Datum der Form dd-mm-yyyy mit dd > 29 und mm = 2 und yyyy ist Schaltjahr
  - ◆ Beispiel: "30-02-2012" → 29.2.2012, 12:00:00
- Tag > 28 (Februar, Säkularjahr)
  - ◆ Testet ob richtig angepasst wird
  - ◆ Voraussetzung: Datum der Form dd-mm-yyyy mit dd > 28 und mm = 2 und yyyy ist Säkularjahr
  - ◆ Beispiel: "29-02-2100" → 28.2.2100, 12:00:00
- Tag > 29 (Februar, Säkularjahr durch 400 teilbar)
  - ◆ Testet ob richtig angepasst wird

- ◆ Voraussetzung: Datum der Form dd-mm-yyyy mit dd > 29 und mm = 2 und yyyy ist Säkularjahr und durch 400 teilbar
- ◆ Beispiel: "30-02-2000" → 29.2.2000, 12:00:00
- Kein Tag angegeben
  - ◆ Testet ob keine Ausnahme auftritt
  - ◆ Voraussetzung: Datum der Form -mm-yyyy
  - ◆ Beispiel: "-01-2010" → null
- Kein Monat angegeben
  - ◆ Testet ob keine Ausnahme auftritt
  - ◆ Voraussetzung: Datum der Form dd--yyyy
  - ◆ Beispiel: "24--2010" → null
- Kein Jahr angegeben
  - ◆ Testet ob keine Ausnahme auftritt
  - ◆ Voraussetzung: Datum der Form dd-mm-
  - ◆ Beispiel: "24-01-" → null
- Kein Jahr angegeben (ohne Bindestrich)
  - ◆ Testet ob keine Ausnahme auftritt
  - ◆ Voraussetzung: Datum der Form dd-mm
  - ◆ Beispiel: "24-01" → null
- Zuviel Bindestriche
  - ◆ Testet ob keine Ausnahme auftritt
  - ◆ Voraussetzung: Datum der Form dd-mm-yyyy-
  - ◆ Beispiel: "24-01-2010-" → null
- Ungültige Zeichen in der Eingabe
  - ◆ Testet ob keine Ausnahme auftritt
  - ◆ Voraussetzung: Eingabe mit ungültigen Zeichen
  - ◆ Beispiel: "24 - 01 - 2010" → null
- Übergabe von null
  - ◆ Testet ob ein leerer String zurückgegeben wird
  - ◆ Voraussetzung: null wird als Eingabe übergeben
  - ◆ Beispiel: null → null
- Übergabe des Leerstrings
  - ◆ Testet ob ein leerer String zurückgegeben wird
  - ◆ Voraussetzung: der Leerstring wird als Eingabe übergeben
  - ◆ Beispiel: "" → null

b) Im SVN-Repository finden Sie im „gruppe/readonly“-Ordner das Projekt „DateParser“, das eine Klasse enthält, die obiges Interface implementiert. Vervollständigen Sie mit *JUnit 4* den `GemaltoDateParserTest`, mit der Sie die Methode `parseDate` in der Klasse `GemaltoParser` testen. Setzen Sie dabei jede in (a) identifizierte Fehlerart in einen Testfall um.

**Hinweis:** Sie können in den Tests die Hilfs-Methode `makeDate(...)` verwenden.

**Bonus:** Welchen Fehler hat die getestete Klasse?

```
import java.util.Date;
import java.util.GregorianCalendar;

import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;
```

```

public class GemaltoDateParserTest {

    protected DateParser systemUnderTest;

    @Before
    public void setUp() {
        systemUnderTest = new GemaltoDateParser();
    }

    /* add your tests here */

    @Test
    public void testStandardInput() {
        assertEquals(makeDate(2010,10,24),
            systemUnderTest.parseDate("24-10-2010"));
    }

    @Test
    public void testLeadingZeros() {
        assertEquals(makeDate(2010,1,4),
            systemUnderTest.parseDate("04-01-2010"));
    }

    @Test
    public void testShort() {
        assertEquals(makeDate(2010,1,4),
            systemUnderTest.parseDate("4-1-2010"));
    }

    @Test
    public void testShortYear() {
        assertEquals(makeDate(2010,1,24),
            systemUnderTest.parseDate("24-1-10"));
    }

    @Test
    public void testYearZero() {
        assertEquals(makeDate(0,1,24),
            systemUnderTest.parseDate("24-01-0000"));
    }

    @Test
    public void testYear9999() {
        assertEquals(makeDate(9999,1,24),
            systemUnderTest.parseDate("24-01-9999"));
    }

    @Test
    public void testMonthBelow1() {
        assertEquals(makeDate(2010,1,24),
            systemUnderTest.parseDate("24-00-2010"));
    }

    @Test
    public void testMonthOver12() {
        assertEquals(makeDate(2010,12,24),
            systemUnderTest.parseDate("24-13-2010"));
    }

    @Test
    public void testDayBelow1() {
        assertEquals(makeDate(2010,1,1),
            systemUnderTest.parseDate("00-01-2010"));
    }
}

```

```

}

@Test
public void testDayOver31() {
    for (int i: new int[]{1,3,5,7,8,10,12})
        assertEquals(makeDate(2010,i,31),systemUnderTest.
            parseDate(String.format("32-%02d-2010",i)));
}

@Test
public void testDayOver30() {
    for (int i: new int[]{4,6,9,11})
        assertEquals(makeDate(2010,i,30),systemUnderTest.
            parseDate(String.format("31-%02d-2010",i)));
}

@Test
public void testFebDayOver28() {
    assertEquals(makeDate(2010,2,28),
        systemUnderTest.parseDate("29-02-2010"));
}

@Test
public void testFebDayOver29LeapYear() {
    assertEquals(makeDate(2012,2,29),
        systemUnderTest.parseDate("30-02-2012"));
}

@Test
public void testFebDayOver28Secular() {
    assertEquals(makeDate(2100,2,28),
        systemUnderTest.parseDate("29-02-2100"));
}

@Test
public void testFebDayOver29SecularBy400() {
    assertEquals(makeDate(2000,2,29),
        systemUnderTest.parseDate("30-02-2000"));
}

@Test
public void testNoDay() {
    assertNull(systemUnderTest.parseDate("-01-2010"));
}

@Test
public void testNoMonth() {
    assertNull(systemUnderTest.parseDate("24--2010"));
}

@Test
public void testNoYear() {
    assertNull(systemUnderTest.parseDate("24-01-"));
}

@Test
public void testNoYearNoDash() {
    assertNull(systemUnderTest.parseDate("24-01"));
}

@Test
public void testAddDashes() {
    assertNull(systemUnderTest.parseDate("24-01-2010-"));
}

```

```

    }

    @Test
    public void testInvalidSpaces() {
        assertNull(systemUnderTest.parseDate("24 - 01 - 2010"));
    }

    @Test
    public void testNull() {
        assertNull(systemUnderTest.parseDate(null));
    }

    @Test
    public void testEmpty() {
        assertNull(systemUnderTest.parseDate(""));
    }

    /* test helper */

    /**
     * Returns a Date object representing 12:00 of a given date
     *
     * @param year the year of the date
     * @param month the month of the date (1-12)
     * @param day the day of the date
     * @return the Date object
     */
    private Date makeDate(int year, int month, int day) {
        return new GregorianCalendar(year,
            month-1, day, 12, 0, 0).getTime();
    }
}

```

Die gegebene Klasse interpretiert zweistellige Jahreszahlen ab 2010 nicht korrekt.

- c) Schreiben Sie nun eine Klasse `MyDateParserTest`, welche von `GemaltoDateParserTest` erbt, aber die `setUp`-Methode so überschreibt, dass in `systemUnderTest` eine Instanz der Klasse `MyDateParser` instantiiert wird. Entwickeln Sie nun die Klasse `MyDateParser`, indem Sie nach jedem Entwicklungsschritt den JUnit-Test `MyDateParser` ausführen, bis alle Testmethoden erfolgreich sind (Zur Dokumentation Ihrer Vorgehensweise sollen Sie nach jeder Modifikation Ihre Klasse in Ihr SVN-Repository einchecken).

```

import org.junit.Before;

public class MyDateParserTest extends GemaltoDateParserTest {

    @Before
    public void setUp() {
        systemUnderTest = new MyDateParser();
    }
}

import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

```

```

public class MyDateParser implements DateParser{

    Pattern parsePattern =
        Pattern.compile("(\\d{1,2})-(\\d{1,2})-((\\d\\d)?\\d\\d)");

    @Override
    public Date parseDate(String input) {
        if (input == null)
            return null;

        Matcher dateMatcher = parsePattern.matcher(input);
        if (dateMatcher.matches()){
            int day = Integer.parseInt(dateMatcher.group(1));
            int month = Integer.parseInt(dateMatcher.group(2));
            int year = Integer.parseInt(dateMatcher.group(3));
            if (dateMatcher.group(4) == null) {
                year += 2000;
            }

            if (month < 1)
                month = 1;
            else if (month > 12)
                month = 12;
            month--;

            if (day < 1)
                day = 1;

            GregorianCalendar date = null;
            for (; ((date == null) ||
                (date.get(Calendar.MONTH) != month) && (day > 0);
                day--) {
                date = new GregorianCalendar
                    (year, month, day, 12, 0, 0);
            }
            if (date != null)
                return date.getTime();
        }
        return null;
    }
}

```