Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III

Übungen zur Vorlesung Softwaretechnologie

- Wintersemester 2019/20 -

Dr. Günter Kniesel

Übungsblatt 12

Zu bearbeiten bis: 10.01.2020, 16 Uhr

Dies ist das letzte zulassungsrelevante Blatt!

Bitte fangen Sie **frühzeitig** mit der Bearbeitung an, damit wir Ihnen bei Bedarf helfen können. Checken Sie die Lösungen zu den Aufgaben bitte in Ihr Repository ein, „Erklärungen“ bitte als Textdatei. Fragen zu Übungsaufgaben respektive zur Vorlesung können Sie auf der Mailingliste swt-tutoren@lists.iai.uni-bonn.de, bzw. swt-vorlesung@lists.iai.uni-bonn.de stellen.

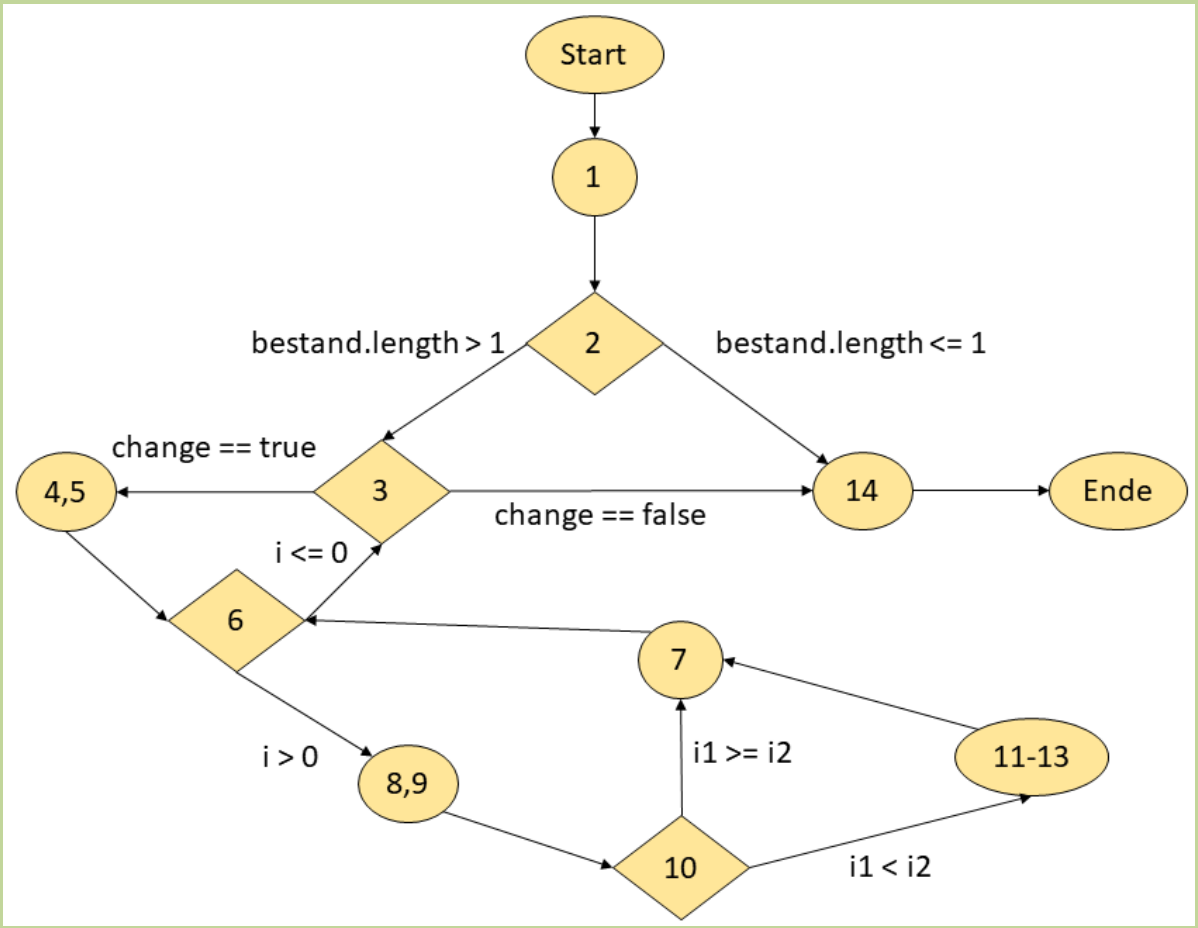
Praktische Aufgaben

Aufgabe 1. *White-Box Test* — 18 Punkte

Gegeben sei die folgende Methode `sortiere`, welche mittels Bubblesort ein Feld von Variablen des Typs `int` sortiert.

```
public int[] sortiere(int[] bestand) {                          //
Anweisungsnr.
      boolean change = true;                                    //  1
      if (bestand.length > 1) {                                 //  2
            while (change) {                                    //  3
                  change = false;                               //  4
                  for (int i = bestand.length - 1;              //  5
                              i > 0;                            //  6
                              i--) {                            //  7
                        int i1 = bestand[i];                    //  8
                        int i2 = bestand[i - 1];                //  9
                        if (i1 < i2) {                          // 10
                              bestand[i] = i2;                  // 11
                              bestand[i - 1] = i1;              // 12
                              change = true;                    // 13
                        }
                  }
            }
      }
      return bestand;                                           // 14
}
```

a) **(8 Punkte)** Entwerfen Sie für die Methode `sortiere` einen Kontrollflussgraphen.



2x0,5P Start- und Endknoten (jeweils mit Kanten) + 6x0,5P Anweisungsknoten (jeweils mit ausgehender Kante) + 4x1P Verzweigungsknoten (mit beiden Kanten und Bedingungen) = 8 Punkte

b) **(3 Punkte)** Geben Sie ein Feld mit Eingabewerten an, das nötig ist, um eine Anweisungsüberdeckung zu erreichen. Schreiben Sie die Reihenfolge auf, in der die Anweisungen getestet werden.

(1 Punkt) [4,3] oder jedes andere Feld mit mindestens zwei Zahlen und für das gilt, dass mindestens zwei Zahlen in der falschen Reihenfolge vorliegen.

(2 Punkte) Die Anweisungsreihenfolge ist (von 1 bis 14):

1 change wird auf true gesetzt
2 if: bestand.length > 1 wird zu true ausgewertet
3 while: change wird zu true ausgewertet
4 change wird auf false gesetzt
5 for: i wird initialisiert (i = 1)
6 for: i > 0 wird zu true ausgewertet
8 i1 wird auf 3 gesetzt
9 i2 wird auf 4 gesetzt
10 if: i1 < i2 wird zu true ausgewertet
11-13 4 und 3 vertauschen; change wird true

7 for: i wird angepasst (i = 0)
6 for: i > 0 wird zu false ausgewertet
3 while: change wird zu true ausgewertet
4 change wird auf false gesetzt
5 for: i wird initialisiert (i = 1)
6 for: i > 0 wird zu true ausgewertet
8 i1 wird auf 4 gesetzt
9 i2 wird auf 3 gesetzt
10 if: i1 < i2 wird zu false ausgewertet
7 for: i wird angepasst (i = 0)
6 for: i > 0 wird zu false ausgewertet
3 while: change wird zu false ausgewertet
14 bestand wird zurückgegeben

c) **(1 Punkt)** Erreichen Sie mit diesem Feld an Eingabewerten auch eine Verzweigungsabdeckung? Begründen Sie kurz Ihre Antwort. Falls ja, geben Sie eine Codemodifikation an, mit der die Verzweigungsabdeckung nicht mehr gegeben wäre. Falls nicht, geben Sie ein weiteres Eingabewerte-Feld an, sodass auch die übrigen Zweige überdeckt werden. Notieren Sie zu diesem neuen Testfall wieder die Reihenfolge der durchgeführten Anweisungen.

Nein. Verzweigungsabdeckung erfordert zusätzlich ein Feld der Größe 1 oder 0 als Eingabe. Die Anweisungsreihenfolge ist dann entsprechend: 1 → 2 → 14

Begründung: Betrachtet man die Anweisungsreihenfolge aus Aufgabenteil a), so ist ersichtlich, dass die Verzweigungsabdeckung für den if-Block in Zeile 10 bereits erfüllt ist. Für eine vollständige Verzweigungsabdeckung muss schließlich noch der else-Teil für den ersten if-Block in Zeile 2 abgedeckt werden. Dies lässt sich mit einem Feld der Größe 1 als zusätzliche Eingabe erreichen.

d) **(5 Punkte)** Wie viele Pfade ohne mehrfache Schleifendurchläufe gibt es? Welche Pfade können vom Programm durchlaufen werden und sollten daher getestet werden? Geben Sie kurze Begründungen an.

- **(3x0,5P + 2x1P)** Es gibt im Graphen 5 Pfade:
    1. 1 → 2 → 14
    2. 1 → 2 → 3 → 14
    3. 1 → 2 → 3 → 4,5 → 6 → 3 → 14
    4. 1 → 2 → 3 → 4,5 → 6 → 8,9 → 10 → 7 → 6 → 3 → 14
    5. 1 → 2 → 3 → 4,5 → 6 → 8,9 → 10 → 11-13 → 7 → 6 → 3 → 14
- **(3x0,5P)** Allerdings wird 2. zur Laufzeit nie durchlaufen: Weil zu diesem Zeitpunkt noch "change == true" gilt, kann die Kante 3 → 14 nicht verfolgt werden.
  Genauso wird 3. nicht durchlaufen, weil der Schritt 6 → 3 nicht möglich ist, da zu diesem Zeitpunkt das Array die Länge 2 hat und somit die Schleifenvariable i >= 1 ist.
  Auch 5. wird nicht durchlaufen, da nach Anweisung 13 "change == true" gilt und ebenfalls die Kante 3 → 14 nicht direkt im Anschluss verfolgt werden kann.

  Beispiel-Eingaben für 1. ist [1], für 4. ist [1,2].

e) **(1 Punkt)** Formulieren Sie ein minimales Programm, für das mindestens zwei verschiedene Testfälle notwendig sind, um eine Anweisungsüberdeckung zu erreichen.

```
public void myMath(int a){
   if (a >= 0)
         a--;
   else
         a++;
}
```

Aufgabe 2. *Black-Box Test* **11+2 Punkte**

Gegeben sei folgende Schnittstelle

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

und folgende Dokumentation der Methode `parseDate(String input)`:

> Erzeugt ein Date-Objekt im gregorianischen Kalender aus einem Datumsstring der Form *Tag-Monat-Jahr*.
>
> - *Tag* und *Monat* können ein- oder zweistellig sein, evtl. mit führender Null.
> - *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 wird der Monatsletzte angenommen.
>
> **Parameter:**
> Der Datumsstring, der interpretiert werden soll
> **Liefert zurück:**
> Das interpretierte Datum (mit dem Uhrzeitwert 12:00:00) oder *null* bei ungültiger

a) **(5 Punkte)** Überlegen Sie, welche Äquivalenzklassen auftreten können (s. Skript, Kapitel 10a. Testen, Seite 20 „Black-box-Tests: Partitionierung“). Geben Sie für fünf Äquivalenzklassen (von korrekten oder falschen Eingaben) einen Eingabestring und das erwartete 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 geparst 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 geparst 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 geparst 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 das nicht in die nächsten beiden Kategorien gehört)
  - 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, nicht durch 400 teilbar)
  - 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) **(6 Punkte)** Im Repository

ssh://gitolite-se@git.iai.uni-bonn.de/swt2019_readonly

finden Sie das Archiv „DateParser.zip“. Darin ist in der Datei `lib/GemaltoDateParser.jar` der ByteCode einer Klasse enthalten, die das Interface `DateParser` implementiert. Im Ordner `tests` finden Sie die noch

unvollständige Klasse `GemaltoDateParserTest` die die Methode `parseDate` aus der Klasse `GemaltoParser` mit *JUnit 4* testet.

Vervollständigen Sie diese Tests indem Sie jede in (a) identifizierte Fehlerart in einen Testfall umsetzen.
Hinweis: Nutzen Sie in den Tests die Hilfs-Methode `makeDate(…)`.

Pro Testmethode 1P (Max. 5) + 1P korrekte JUnit Annotationen

```
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 testAddDashs() {
		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();
	}
}
```

c) **(2 Bonuspunkte)** Welchen Fehler hat die getestete Klasse?

Die gegebene Klasse interpretiert zweistellige Jahreszahlen ab 2010 nicht korrekt.

Theoretische Aufgaben

Aufgabe 3. *Objektentwurf* — 7 Punkte

a) **(1,5 Punkte)** Nennen Sie drei UML-Konzepte, die nicht in Java realisierbar sind.

- Multiple Vererbung
- Multiple Klassifikation
- Dynamische Klassifikation

b) **(1 Punkt)** Wie können Sie zustandsbasiertes Verhalten mit Hilfe eines Design Patterns modellieren?

State Entwurfsmuster.

c) **(1,5 Punkte)** Erklären Sie genau, was *Forwarding*, *Subtyping* und *Overriding* bedeuten.

- Forwarding: Kindobjekt leitet Nachricht an Elternobjekt weiter
- Subtyping: Kindobjekt bietet volles Interface des Elternobjekts
- Overriding: Nutze Methoden des Kindobjekts statt derer des Elternobjekts

d) **(3 Punkte)** Welches Entwurfsmuster bietet sich an, um Werte abgeleiteter Attribute konsistent mit den jeweiligen Basiswerten zu halten? Erklären Sie an einem Beispiel, wie Sie das Entwurfsmuster anwenden würden.
Observer Entwurfsmuster (1 Punkt)
Das Objekt, das das abgeleitete Attribut enthält, ist Observer der Objekte, die die Basiswerte enthalten. Siehe Folie 09-09: „Umsetzung abgeleiteter Attribute“
(2 Punkte)

∑ 36 Punkte