Gruppe A	PRÜFUNG AUS "SEMISTRUKTURIERTE DATEN" **181.135**		19. 11. 2009
Kennnr.	Matrikelnr.	Familienname	Vorname

Arbeitszeit: 120 Minuten. Aufgaben sind auf den Angabeblättern zu lösen; Zusatzblätter werden nicht gewertet.
Bitte tragen Sie Namen, Studienkennzahl und Matrikelnummer ein. Halten Sie Ihren Studentenausweis bereit.

Aufgabe 1: (9)

Betrachten Sie die folgende DTD **test.dtd**:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT A (B*,C*)>
<!ELEMENT B (B?,C,C?)>
<!ELEMENT C (#PCDATA|D|E)*>
<!ATTLIST C F IDREF #IMPLIED>
<!ELEMENT D EMPTY>
<!ELEMENT E (#PCDATA)>
<!ATTLIST E P ID #REQUIRED>
```

Betrachten Sie weiters die XML-Dateien, die unten angeführt sind.
Hinweise:

- Gehen Sie davon aus, dass allen folgenden Dateien die Zeile
  `<!DOCTYPE A SYSTEM "test.dtd">`
  vorangestellt ist.
- Sie können auch davon ausgehen, dass alle folgenden XML-Dateien wohlgeformt sind. Es geht also lediglich darum, ihre Gültigkeit bezüglich **test.dtd** zu entscheiden.

Kreuzen Sie an, welche der folgenden XML-Dateien gültig bezüglich **test.dtd** sind.

1. `<A><C F="n1"><E P="n1"/><E P="n2">test</E></C></A>` gültig ◯ ungültig ◯
2. `<A><C>cde<E P="n1">fgh</E></C></A>` gültig ◯ ungültig ◯
3. `<A><B><C></C></B></A>` gültig ◯ ungültig ◯
4. `<A><C F="n1">test</C><C><E P="n1"/></C></A>` gültig ◯ ungültig ◯
5. `<A><B><C/><C/><C/></B></A>` gültig ◯ ungültig ◯
6. `<A><B><B><C/></B></B></A>` gültig ◯ ungültig ◯

(Pro korrekter Antwort 1.5 Punkte, **pro inkorrekter Antwort -1.5 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

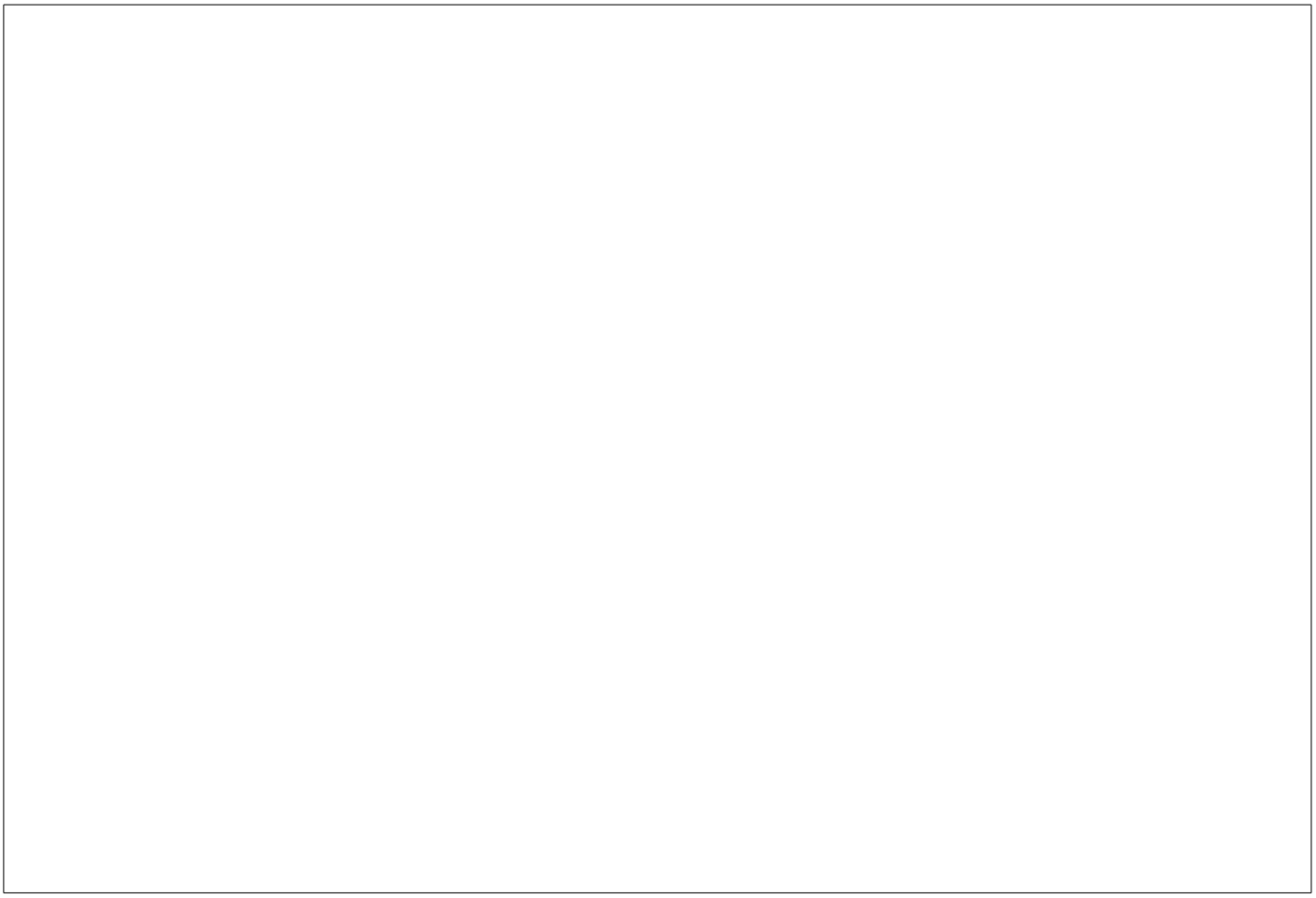
Aufgabe 2: (9)

Nehmen Sie an, dass folgender Java Code (links) auf das angegebene XML-Dokument (rechts) ausgeführt wird. Zeichnen Sie den resultierenden DOM-Baum *nach* der Ausführung des Codes auf.

Bemerkung: Nehmen Sie an, dass die angegebene DTD eventuelle "versteckte" Textknoten verhindert. Dh. Sie brauchen sich nicht um leere Textknoten zwischen Elementen kümmern.

```
class Exam {
  public static void main() {
    Document documentNode; // DOCUMENT Node
    // Gegeben: Dokument lesen etc...
    secret(documentNode);
    // Aufgabe: DOM Baum zeichnen
  }
  private static void secret(Node n) {
    NodeList nl = n.getChildNodes();
    for(int i=0; i < nl.getLength(); i++) {
      Node child = nl.item(i);
      String x = child.getNodeValue();
      if(x != null) {
        child.setNodeValue("A");
      }
      secret(child);
    }
  }
}
```

```
<?xml version="1.0"?>
<!DOCTYPE tuwien SYSTEM "secret.dtd">
<!--Author: ssd@dbai.tuwien.ac.at-->
<?exam this is difficult?>
<tuwien>
  <faculty>
    <name>Informatik</name>
  </faculty>
</tuwien>
```



Aufgabe 3: (9)

Betrachten Sie die folgende XML- Datei **ns.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<A xmlns:ns1="uri1">
  <ns1:B xmlns="uri2" xmlns:ns1="uri3">
    <D attr1="X"/>
  </ns1:B>
  <ns1:C>
    <E ns1:attr2="Y"/>
  </ns1:C>
</A>
```

Kreuzen Sie an, ob die folgenden Aussagen für die Datei **ns.xml** wahr oder falsch sind.

1. Das Element `A` liegt im Namespace `uri1`. wahr ◯ falsch ◯
2. Das Element `ns1:B` liegt im Namespace `uri2`. wahr ◯ falsch ◯
3. Das Element `D` liegt im Namespace `uri2`. wahr ◯ falsch ◯
4. Das Element `E` liegt im Namespace `uri2`. wahr ◯ falsch ◯
5. Das Attribut `attr1` liegt im leeren Namespace. wahr ◯ falsch ◯
6. Das Attribut `attr2` liegt im Namespace `uri1`. wahr ◯ falsch ◯

(Pro korrekter Antwort 1.5 Punkte, **pro inkorrekter Antwort -1.5 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

Aufgabe 4: (12)

Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind.

1. `<paragraph><text/>multiple choice</text></paragraph>` ist wohlgeformt. wahr ◯ falsch ◯
2. Die DTD Elementdeklaration `(A*|B*|C*)` ist gleichwertig mit `(A|B|C)*`. wahr ◯ falsch ◯
3. Namespaces können in DTDs unabhängig vom Präfix definiert werden. wahr ◯ falsch ◯
4. DOM erlaubt wahlfreien Zugriff auf das gesamte XML Dokument. wahr ◯ falsch ◯
5. Mit einem XMLFilter ist es unter anderem möglich, den Namespace eines Elements zu ändern. wahr ◯ falsch ◯
6. Der XPath-Ausdruck `//oper` ist die Kurzschreibweise des XPath-Ausdrucks `/descendant-or-self::oper` wahr ◯ falsch ◯
7. Mit XSLT ist es möglich, XML Dokumente in beliebige andere Dokumente (zB Text, HTML etc) zu transformieren. wahr ◯ falsch ◯
8. Ein XSLT-Stylesheet muss ein wohlgeformtes XML-Dokument sein. wahr ◯ falsch ◯

(Pro korrekter Antwort 1.5 Punkte, **pro inkorrekter Antwort -1.5 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

Aufgabe 5: (8)

Vervollständigen Sie die DTD **oper.dtd**, sodass das XML-Dokument **oper.xml** (siehe Anhang) bezüglich dieser DTD gültig ist. Berücksichtigen Sie beim Erstellen der DTD folgende Punkte:

- Die Reihenfolge der Elemente soll wie in **oper.xml** angegeben fixiert werden.
- Alle Attribute sind verpflichtend.
- Das Wurzelelement (`oper`) beinhaltet jeweils genau ein Element `werke`, `inszenierungen` und `wertungen`.
- `werke` enthält ein oder mehrere `werk` Elemente.
- `werk` enthält genau ein `komponist`, `name` und `urauffuehrung` Element. Das Attribut `id` soll global eindeutig sein.
- `inszenierungen` enthält ein oder mehrere `inszenierung` Elemente.
- `inszenierung` enthält jeweils ein `regie`, `ort` und `premiere` Element. Das Attribut `id` von `inszenierung` soll global eindeutig sein, und das Attribut `werk` von `inszenierung` soll vom Typ `IDREF` sein.
- `wertungen` enthält beliebig viele `bewertung` Elemente.
- `bewertung` ist ein leeres Element und hat zwei Attribute `wert` und `inszenierung`. Stellen Sie sicher dass das Attribut `wert` nur die Werte 1 bis 5 annehmen kann. Das Attribut `inszenierung` soll vom Typ `IDREF` sein.

Datei **oper.dtd**:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT komponist (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT urauffuehrung (#PCDATA)>
<!ELEMENT regie (#PCDATA)>
<!ELEMENT ort (#PCDATA)>
<!ELEMENT premiere (#PCDATA)>
```

Aufgabe 6: (12)

Betrachten Sie die folgenden drei XSLT-Stylesheets. Geben Sie jeweils den Output an, den das entsprechende Stylesheet angewandt auf **oper.xml** (siehe Anhang) liefert. Sie brauchen sich dabei nicht um Whitespaces etc. kümmern.

Anmerkung: Pro Teilaufgabe sind jeweils 4 Punkte erreichbar.

Nehmen Sie an dass jede Datei mit einem korrekten Header versehen ist. zB

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Datei **query1.xsl**:

```
<xsl:output method="text" encoding="UTF-8" version="1.0" />

<xsl:template match="/">
  <xsl:for-each select="/oper/inszenierungen/*">
    <xsl:variable name="id" select="./@id"/>
    <xsl:variable name="werk" select="./@werk"/>
    Werk: <xsl:value-of select="//werk[@id=$werk]/name"/>
    Regie: <xsl:value-of select="regie"/>
    <xsl:if test="//bewertung[@inszenierung=$id]">
      Bewertung: <xsl:value-of select="//bewertung[@inszenierung=$id]/@wert"/>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Geben Sie hier den Output von **query1.xsl** angewandt auf **oper.xml** an:

Datei **query2.xsl**:

```
<xsl:output method="xml" indent="yes" encoding="UTF-8" version="1.0" />

<xsl:template match="/">
  <xsl:for-each select="//werk[komponist='Puccini']">
    <xsl:if test="position()=last()">
       <xsl:copy-of select="."/>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Geben Sie hier den Output von **query2.xsl** angewandt auf **oper.xml** an:

Datei **query3.xsl**:

```
<xsl:output method="text" encoding="UTF-8" version="1.0" />

<xsl:template match="/oper/werke">
  <!-- empty -->
</xsl:template>
<xsl:template match="/oper/werke/werk">
  <xsl:copy-of select="."/>
</xsl:template>
<xsl:template match="/oper/inszenierungen">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="/oper/inszenierungen/inszenierung">
  Inszenierung: <xsl:value-of select="./premiere"/>
</xsl:template>
<xsl:template match="/oper/wertungen">
  <xsl:apply-templates select="./bewertung[@wert > 10]"/>
</xsl:template>
</xsl:stylesheet>
```

Geben Sie hier den Output von **query3.xsl** angewandt auf **oper.xml** an:

Aufgabe 7: (8)

Schreiben Sie einen **XMLFilter** der folgendes XML-Format in ein gültiges XML-Dokument (bezgl. `oper.dtd`) transformiert. Sie müssen also den Elementnamen korrigieren. Beispiel (Input → Output):

```
<oper>
  <werke><!--...--></werke>
  <inszenierungen><!--...--></inszenierungen>
  <wertungen>
    <wert inszenierung="i1" wert="2"/>
    <wert inszenierung="i2" wert="4"/>
    <wert inszenierung="i3" wert="5"/>
  </wertungen>
</oper>
```

```
<oper>
  <werke><!--...--></werke>
  <inszenierungen><!--...--></inszenierungen>
  <wertungen>
    <bewertung inszenierung="i1" wert="2"/>
    <bewertung inszenierung="i2" wert="4"/>
    <bewertung inszenierung="i3" wert="5"/>
  </wertungen>
</oper>
```

Tipp: Die Klasse `OperKorrektur` erweitert `XMLFilterImpl`, dh. per default werden alle Events unverändert durchgereicht. Überschreiben Sie die relevanten Methoden. Nehmen Sie an, dass Namespace-Parsing aktiviert ist. Dh. Sie brauchen sich nur um den lokalen Namen (localName) von Elementen kümmern.

```
class OperKorrektur extends XMLFilterImpl {









}
```

Aufgabe 8: (8)

Schreiben Sie XPath-Anfragen für folgende Aufgabenstellungen. Zu jeder Abfrage ist ein Beispiel mit der erwarteten Ausgabe (bezgl. `oper.xml`) angegeben. Die Abfragen sollen auf allen Dokumenten, die gültig bezgl. `oper.dtd` sind, funktionieren.

1. Geben Sie den Namen aller Werke aus, die nach 1900 uraufgeführt wurden.

```
<name>Turandot</name>
```

2. Geben Sie alle Namen der Werke aus, für die eine Bewertung einer Inszenierung vorliegt.

```
<name>Tosca</name>
<name>Don Giovanni</name>
```

3. Geben Sie jene Orte einer Inszenierung aus, für die **keine** Bewertung vorliegt.

```
<ort>Wiener Volksoper</ort>
```

4. Geben Sie das letzte Werk (Document-Order) von Puccini in unserer XML-Datei aus.

```
<werk id="w3">
  <komponist>Puccini</komponist>
  <name>Turandot</name>
  <urauffuehrung>1926</urauffuehrung>
</werk>
```

Sie können diese Seite abtrennen!

oper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE oper SYSTEM "oper.dtd">
<oper>
  <werke>
    <werk id="w1">
      <komponist>Verdi</komponist>
      <name>La Traviata</name>
      <urauffuehrung>1853</urauffuehrung>
    </werk>
    <werk id="w2">
      <komponist>Puccini</komponist>
      <name>Tosca</name>
      <urauffuehrung>1900</urauffuehrung>
    </werk>
    <werk id="w3">
      <komponist>Puccini</komponist>
      <name>Turandot</name>
      <urauffuehrung>1926</urauffuehrung>
    </werk>
    <werk id="w4">
      <komponist>Mozart</komponist>
      <name>Don Giovanni</name>
      <urauffuehrung>1787</urauffuehrung>
    </werk>
  </werke>
  <inszenierungen>
    <inszenierung id="i1" werk="w4">
      <regie>Willy Decker</regie>
      <ort>Sächsische Staatsoper</ort>
      <premiere>1993</premiere>
    </inszenierung>
    <inszenierung id="i2" werk="w1">
      <regie>Hans Gratzer</regie>
      <ort>Wiener Volksoper</ort>
      <premiere>2007</premiere>
    </inszenierung>
    <inszenierung id="i3" werk="w2">
      <regie>Margarethe Wallmann</regie>
      <ort>Wiener Staatsoper</ort>
      <premiere>1957</premiere>
    </inszenierung>
  </inszenierungen>
  <wertungen>
    <bewertung inszenierung="i1" wert="2"/>
    <bewertung inszenierung="i3" wert="5"/>
  </wertungen>
</oper>
```

Gesamtpunkte: 75