PRÜFUNG IN "SEMI-STRUKTURIERTE DATEN" **184.705**			23. 10. 2017
Kennnr.	Matrikelnr.	Familienname	Vorname

Arbeitszeit: 100 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: (12)

Betrachten Sie folgende XML Schema Datei **test.xsd**:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="A">
        <xsd:complexType mixed="false" >
            <xsd:choice minOccurs="2"  maxOccurs="2">
                <xsd:element name="B" type="xsd:boolean" maxOccurs="2" fixed="true"/>
                <xsd:element name="C" type="typeC">
                    <xsd:unique name="constraint1">
                        <xsd:selector xpath="D"/>
                        <xsd:field xpath="@id"/>
                    </xsd:unique>
                </xsd:element>
            </xsd:choice>
        </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="typeC" mixed="false">
         <xsd:sequence>
                <xsd:element name="D" type="typeD" minOccurs="0" maxOccurs="10"/>
          </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="typeD" mixed="true">
        <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
 </xsd:schema>
```

Betrachten Sie weiters die acht verschiedenen XML-Dateien, die unten angeführt sind.

Sie können davon ausgehen, dass alle folgenden XML-Dateien wohlgeformt sind. Es geht also lediglich darum, ihre Gültigkeit bezüglich test.xsd zu entscheiden.

Kreuzen Sie an, welche der folgenden xml-Dateien gültig bezüglich test.xsd sind.

1. `<A><B>true</B><B>true</B></A>` valid ⊗ invalid ◯
2. `<A><B>true</B><B>true</B><B>true</B></A>` valid ⊗ invalid ◯
3. `<A><B>true</B><C/><B>true</B></A>` valid ◯ invalid ⊗
4. `<C/>` valid ◯ invalid ⊗
5. `<A>SSD:<C/><B>false</B></A>` valid ◯ invalid ⊗
6. `<A><C><D id="1">SSD</D></C><C><D id="1"></D></C></A>` valid ⊗ invalid ◯
7. `<A><C><D id="1"></D><D/></C><C/></A>` valid ⊗ invalid ◯
8. `<A><C><D id="4"/><D id="4"/></C><C/></A>` valid ◯ invalid ⊗

(Für jede korrekte Antwort 1.5 Punkte, **für jede falsche Antwort -1.5 Punkte**, unbeanwortete Fragen 0 Punkt, Insgesamt nicht weniger als 0 Punkte)

Aufgabe 2: (15)
Entscheiden Sie, ob die folgenden Aussagen wahr oder falsch sind.

1.	DOM und SAX sind Standards für Event-Based XML-Parser.	wahr ◯ falsch ⊗
2.	DTDs sind immer wohlgeformte XML Dokumente.	wahr ◯ falsch ⊗
3.	DTDs und XML Schema sind gleichmächtig.	wahr ◯ falsch ⊗
4.	Die DOM API kommt nur bei XHTML Dokumenten zum Einsatz.	wahr ◯ falsch ⊗
5.	Strukturierte Daten sind ein Spezialfall von semi-strukturierten Daten.	wahr ⊗ falsch ◯
6.	XML Dokumente sind das einzige Format um semi-strukturierten Daten zu speichern.	wahr ◯ falsch ⊗
7.	Event-based Parser benötigen nur konstant viel Speicher, d.h. der Speicherbedarf ist unabhängig von der Größe des geparsten XML Dokuments.	wahr ⊗ falsch ◯
8.	Das übliche Datenmodell für semi-strukturierte Daten sind Bäume.	wahr ⊗ falsch ◯
9.	Namespace URIs sind immer gültige XML Namen.	wahr ◯ falsch ⊗
10.	Nur Elemente ohne Prefix sind im Default Namespace.	wahr ◯ falsch ⊗

(Für jede korrekte Antwort 1.5 Punkte, **für jede falsche Antwort -1.5 Punkte**, unbeanwortete Fragen 0 Punkt, Insgesamt nicht weniger als 0 Punkte)

Die folgenden Aufgaben 3 – 7 beziehen sich auf das XML-Dokument boatrent.xml, das Sie auf der letzten Seite dieser Prüfungsangabe finden.

Aufgabe 3: (12)

Vervollständigen Sie das DTD Dokument **boatrent.dtd**, sodass XML-Dokumente in der Gestalt von **boatrent.xml** (siehe Anhang) bezüglich dieser DTD gültig sind. Berücksichtigen Sie beim Erstellen der DTD folgende Punkte:

- Das `boats_for_rent` Element enthält beliebig viele `boat` Elemente.
- Das `customers` Element enthält beliebig viele `customer` Elemente.
- `boat` Elemente enthalten ein `name` Element, ein `type` Element, möglicherweise ein `subtype` Element, ein `dock` Element, ein `capacity` Element, wenn es zu vermieten ist ein `price_hour` Element, und eventuell ein `required_licence` Element (in dieser Reihenfolge).
- `costumer` Elemente enthalten ein `name` Element und beliebig viele `licence`, `boat` und `rent` Elemente (in dieser Reihenfolge).
- Jedes `boat` und jedes `costumer` Element hat ein Attribut `id` das einen eindeutigen Wert hat.
- `rent` Elemente haben drei Attributte `boatid`, `start`, `duration` wobei das Letzte optional ist. Die `boatid` verweist auf die `id` eines `boat` Elements.
- Wenn nicht angegeben treffen Sie plausible Annahmen über die Typen von Elementen.

File **boatrent.dtd**:

```
<!ELEMENT boatrent (boats_for_rent, customers)>

<!ELEMENT boats_for_rent (boat*)>

<!ELEMENT customers (customer*)>

<!ELEMENT boat (name,type,subtype?,dock,capacity,price_hour?,required_licence?)>
<!ATTLIST boat id ID #REQUIRED>

<!ELEMENT customer (name,licence*,boat*,rent*)>
<!ATTLIST customer id ID #REQUIRED>

<!ELEMENT rent EMPTY>
<!ATTLIST rent boatid IDREF #REQUIRED start CDATA #REQUIRED duration CDATA #IMPLIED>

<!ELEMENT name (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT subtype (#PCDATA)>
<!ELEMENT dock (#PCDATA)>
<!ELEMENT capacity (#PCDATA)>
<!ELEMENT price_hour (#PCDATA)>
<!ELEMENT required_licence (#PCDATA)>
<!ELEMENT licence (#PCDATA)>
```

Aufgabe 4: (10)

Betrachten Sie die folgenden XPath-Abfragen angewandt auf das Dokument **boatrent.xml** (siehe Anhang).

- Falls der angegebene XPath Ausdruck keine Knoten selektiert, notieren Sie im entsprechenden Feld "leere Ausgabe".
- Falls als Ergebnis eine Zahl selektiert wird (count, sum, ...), geben Sie diese Zahl an.

Geben Sie nun die entsprechenden Ausgaben der folgenden XPath-Abfragen an.

```
sum(//boats_for_rent//capacity)
```

```
13
```

```
//boat[@id=//rent/@boatid][last()]/name
```

```
<name>Bounty</name>
```

```
//boat[required_licence][2]/type
```

```
<type>sailboat</type>
```

```
//boat[3][required_licence]/type
```

```
<type>motorboat</type>
```

```
//boats_for_rent/boat[not(@id=//@boatid)]/name/text()
```

```
Vancouver
Eva
```

Aufgabe 5: (10)

Schreiben Sie eine XQuery Abfrage um folgende Aufgabenstellung zu lösen:

> Es soll ein XML Dokument mit der Wurzel `boats` erzeugt werden. In diesem Dokument sollen alle Boote ausgegeben werden, die zumindest einmal verliehen wurden. Geben Sie die Boote sortiert nach deren Namen als `boat` Elemente aus. Für jedes Boot geben Sie auch ein Attribut `id` an, dass die ID des Boots enthält. Zusätzlich legen Sie ein Kindelement `name` mit dem Namen des Boots und ein Kindelement `revenue` mit dem Umsatz des Boots an. Der Umsatz wird berechnet aus der Gesamtsumme an verliehenen Stunden (Summe von `duration`) multipliziert mit dem Preis pro Stunde des Boots.

Ihre Abfrage soll, angewandt auf **boatrent.xml**, folgende Ausgabe liefern:

```
<boats>
  <boat id="b3">
    <name>Bounty</name>
    <revenue>90</revenue>
  </boat>
  <boat id="b1">
    <name>Voyager</name>
    <revenue>168</revenue>
  </boat>
</boats>
```

Geben Sie nun die XQuery Abfrage an:

```
<boats>
{
    for $b in //boat
    where count(//rent[@boatid=$b/@id]) > 0
    order by $b/name ascending
    return
        <boat id="{$b/@id}">
            {$b/name}
            <revenue>{sum(//rent[@boatid=$b/@id]/@duration)*$b/price_hour}</revenue>
        </boat>
}
</boats>
```

Aufgabe 6: (10)

Erstellen Sie ein XSLT-Stylesheet **boatrent.xsl**, das angewandt auf Dokumente der Gestalt **boatrent.xml** folgende Ausgabe liefert:

- Ausgegeben werden soll ein XML Dokument.
- Das Wurzelelement heißt `possible_boats`.
- Für jeden Kunden soll ein `customer`-Element mit dessen Namen im `name` Attribut ausgegeben werden.
- Pro Kunden sollen alle Boote (`boot`-Element mit Bootsnamen als Inhalt) ausgegeben werden, mit denen der Kunde fahren darf. Dies sind folgende:
  - Des Kunden eigene Boote (`boat`-Element des `customer`-Elements).
  - Boote die keine Lizenz benötigen (`boat`-Elemente in `boats_for_rent` ohne `required_licence`).
  - Boote für die der Kunde die Lizenz hat (`boat`-Elemente in `boats_for_rent` mit `required_licence` gleich eines `licence`-Element des Kunden).

Betrachten Sie dazu folgende Ausgabe, die ihr XSLT-Stylesheet **boatrent.xsl** angewandt auf **boatrent.xml** (siehe Anhang) produzieren soll:

```
<?xml version="1.0" encoding="UTF-8"?>
<possible_boats>
    <customer name="Alfred Nobel">
        <boat>Vasa</boat>
        <boat>Voyager</boat>
        <boat>Vancouver</boat>
    </customer>
    <customer name="Donald Duck">
        <boat>Bounty</boat>
        <boat>Voyager</boat>
        <boat>Vancouver</boat>
    </customer>
    <customer name="James Bond">
        <boat>Queen Elisabeth</boat>
        <boat>Eva</boat>
        <boat>Voyager</boat>
        <boat>Vancouver</boat>
    </customer>
</possible_boats>
```

Vervollständigen Sie hier das XSLT-Stylesheet **boatrent.xsl**. Sie brauchen sich nicht um Whitespaces etc. zu kümmern.

File **boatrent.xsl**:

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

    <xsl:template match="boatrent">
        <possible_boats>
        <xsl:apply-templates select="//customer" />
        </possible_boats>
    </xsl:template>

    <xsl:template match="customer">
        <customer name="{name}">
           <xsl:apply-templates select="boat" />
           <xsl:apply-templates select="licence" />
           <xsl:apply-templates select="//boats_for_rent/boat[not(required_licence)]" />
        </customer>
    </xsl:template>

    <xsl:template match="licence">
        <xsl:variable name="lic" select="text()" />
        <xsl:apply-templates select="//boats_for_rent/boat[required_licence=$lic]" />
    </xsl:template>

    <xsl:template match="boat">
        <boat><xsl:value-of select="name" /></boat>
    </xsl:template>

</xsl:stylesheet>
```

Aufgabe 7: (6)

Betrachten Sie die folgende Java Klasse und geben Sie die Ausgabe der Klasse an, wenn als Input die Datei **boatrent.xml** verwendet wird.

```
public class RunSAX extends DefaultHandler {
    private String eleText;
    int i = 0;
    HashMap<String, Integer> cnt = new HashMap<String, Integer>();

    @Override
      public void characters(char[] text, int start, int length)
                  throws SAXException {
          eleText = new String(text, start, length);
      }

    @Override
    public void startElement(String namespaceURI, String localName, String qName, Attributes atts)
            throws SAXException {
        if ("boat".equals(localName) || "type".equals(localName) || "subtype".equals(localName))
            i++;
    }

    @Override
    public void endElement(String namespaceURI, String localName, String qName) throws SAXException {
        if ( i == 2 ) {
            if (cnt.containsKey(eleText))
                cnt.replace(eleText, cnt.get(eleText)+1);
            else
                cnt.put(eleText, 1);
        }

        if ("boat".equals(localName) || "type".equals(localName) || "subtype".equals(localName))
            i--;
    }

    @Override
    public void endDocument() {
        for (Map.Entry<String, Integer> entry : cnt.entrySet())
            System.out.println(entry.getKey() + ": " + entry.getValue());
    }

    public static void main(String[] args) throws Exception {
        String input = args[0];

        InputSource source = new InputSource(new FileInputStream(input));
        XMLReader xr = XMLReaderFactory.createXMLReader();
        RunSAX rs = new RunSAX();
        xr.setContentHandler(rs);

        xr.parse(source);
    }
}
```

Geben Sie hier die Ausgabe an:

```
Optimist: 1
sailboat: 3
motorboat: 1
canoe: 2
Laser: 1
canadian: 1
kayak: 1
```

Total points: 75

Sie können diese Seite abtrennen!

File **boatrent.xml**:

```
<boatrent>
 <boats_for_rent>
    <boat id="b1">
        <name>Voyager</name>
        <type>canoe</type><subtype>kayak</subtype>
        <dock>1</dock>
        <capacity>2</capacity><price_hour>14</price_hour>
    </boat>
    <boat id="b2">
        <name>Vancouver</name>
        <type>canoe</type> <subtype>canadian</subtype>
        <dock>2</dock>
        <capacity>3</capacity> <price_hour>15</price_hour>
    </boat>
    <boat id="b3">
        <name>Bounty</name>
        <type>motorboat</type>
        <dock>3</dock>
        <capacity>6</capacity>
        <price_hour>30</price_hour>
        <required_licence>motorboat</required_licence>
    </boat>
    <boat id="b4">
        <name>Eva</name>
        <type>sailboat</type><subtype>Laser</subtype>
        <dock>4</dock>
        <capacity>2</capacity><price_hour>20</price_hour>
        <required_licence>sailboat</required_licence>
    </boat>
 </boats_for_rent>
 <customers>
    <customer id="c1">
        <name>Alfred Nobel</name>
        <boat id="b5">
            <name>Vasa</name>
            <type>sailboat</type><subtype>Optimist</subtype>
            <dock>42</dock>
            <capacity>1</capacity>
        </boat>
    </customer>
    <customer id="c2">
        <name>Donald Duck</name>
        <licence>motorboat</licence>
        <rent boatid="b1" start="5/9/2017 8:00" duration="4"/>
        <rent boatid="b3" start="23/9/2017 9:00" duration="3"/>
    </customer>
    <customer id="c007">
        <name>James Bond</name>
        <licence>Licence to Kill</licence>
        <licence>sailboat</licence>
        <boat id="b8">
            <name>Queen Elisabeth</name>
            <type>sailboat</type>
            <dock>11</dock>
            <capacity>4</capacity>
        </boat>
        <rent boatid="b1" start="22/9/2017 8:00" duration="8"/>
        <rent boatid="b1" start="23/10/2017 17:00"/>
    </customer>
 </customers>
</boatrent>
```