PRÜFUNG IN "SEMI-STRUKTURIERTE DATEN" **184.705**			25. 6. 2019
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**:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:element name="W" type="newtype"/>

  <xsd:complexType name="newtype">
       <xsd:choice>
          <xsd:sequence>
             <xsd:element name="Z" type="xsd:NMTOKEN"/>
             <xsd:element name="W" minOccurs="0" maxOccurs="2" type="newtype"/>
          </xsd:sequence>
          <xsd:sequence>
             <xsd:element name="W" type="newtype"/>
             <xsd:element name="Z" type="xsd:int"/>
          </xsd:sequence>
       </xsd:choice>
  </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. `<W><Z>SSD</Z></W>` valid ○ invalid ○
2. `<W><Z>Einstein</Z><W><Z>A</Z></W></W>` valid ○ invalid ○
3. `<W><Z>Emmy Noether</Z></W>` valid ○ invalid ○
4. `<W><Z>Albert</Z><Z>1</Z></W>` valid ○ invalid ○
5. `<W><W><W><Z>ADBS</Z></W><Z>007</Z></W></W>` valid ○ invalid ○
6. `<W><W><W><Z>ADBS</Z></W><Z>100</Z></W><Z>42</Z></W>` 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)
Beantworten Sie, die folgenden Fragen kurz und bündig (Für jede korrekte Antwort 1.5 Punkte).

1. Erläutern Sie kurz den Unterschied zwischen `let` und `for` statements in XQuery.

2. Wie verhält sich der Speicherbedarf von Tree-based Parsern wie DOM zur Größe des geparsten XML Dokuments?

3. Was macht das Default-Template in XSLT für Elemente?

4. Wie geht XSLT mit der Situation um wenn mehrere Templates auf ein Element matchen?

5. Welches Datenmodell verwenden wir für semistrukturierte Daten?

6. Auf welche Teile eines XML Dokuments wird der Default Namespace angewendet?

7. Geben Sie einen Formalismus (ein Format) an um semistrukturierte Daten zu speichern:

8. Wir haben verschiedene Sprachen um Schemata zu definieren kennen gelernt. Welche hat die größte Ausdrucksstärke?

9. Zu welchem Zweck werden Namespaces verwendet?

10. Wieso können URIs nicht direkt als prefix für XML Elemente/Attribute verwendet werden?

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

Aufgabe 3: (12)

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

- Ein `product` Element enthält ein `productname` Element, möglicherweise ein `description` Element, und beliebig viele `category` Elemente (in dieser Reihenfolge). Zusätzliche enthält es immer zwei Attribute: eine eindeutige id `ID` und ein Attribute auctionEnd.
- Das `description` Element enthält eine textuelle Beschreibung und kann Referenzen mittels `a` Elementen enthalten. `a` Elemente enthalten immer ein Attribut `href`.
- Im `categories` Element können beliebig viele `category` Elemente gespeichert sein.
- `category` Elemente tretten in zwei Formen auf. Als Kind des `categories` Elements enthält es eine Bezeichnung und ein Attribut `id` als eindeutige id. Als Kind eines `product` Elements ist es leer und hat ein Attribut `ref` das auf die id eines unter `categories` gespeicherten `category` Elements verweist.
- Das `user` Element enthält beliebig viele `user` Elemente. Jedes user `Element` enthält entweder ein `fullname` oder ein `firstname` und ein `lastname` Element. Zusätzlich gibt es immer in eindeutiges `username` Attribut.
- Das `bids` Element enthält beliebg viele `bidproduct` Elemente. Jedes `bidproduct` enthält mindestens ein `bid` Element und eine Attribut `id` das auf ein `product` Element verweist. `bid` Elemente speichern eine ganze Zahl und ein Attribut `user` das auf den usernamen eines users verweisen.
- Wenn nicht angegeben treffen Sie plausible Annahmen über Typen von Attributen und Elementen.

File **auctions.dtd**:

```
<!ELEMENT auctions (products, categories, users, bids)>
```

Aufgabe 4: (10)

Betrachten Sie die folgenden XPath-Abfragen angewandt auf das Dokument **auctions.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 entsprechende Ausgaben der folgenden XPath-Abfragen an.

```
count(//categories/category)
```

```
//product[./category/@ref=//categories/category[text()="book"]/@id]/@id
```

```
//product[category/@ref="cat1"][2]/productname
```

```
//product[2][category/@ref="cat1"]/productname
```

```
//bids/bidproduct[sum(bid)>499]/@id
```

Aufgabe 5: (10)

Ergänzen Sie die Java Klasse auf dieser und der nächsten Seite so, dass sie beim Aufruf, mit XML files die dem Schema der gegebenen **auctions.xml** folgen, den folgenden Output erzeugt:

- Für jeden User soll eine Zeile ausgegeben werden in der die **Summe** aller Gebote des Users zusammen mit dem `username` steht.
- Halten Sie sich bei der Formatierung an den Beispiel Output.
- Hat ein user nie ein Gebot abgegeben, so muss auch keine Zeile für ihn ausgegeben werden.

Beispiel Output beim Aufruf mit der gegebenen **auctions.xml**:

```
Homer bid 200 in total
Bart bid 1300 in total
Lisa bid 410 in total
```

(Reihenfolge der user undefiniert)

Vervollständigen Sie den folgenden Code:

```
public class RunSAX extends DefaultHandler {
  String eleText;









public void characters(char[] text, int start, int length) throws SAXException {
  eleText = new String(text, start, length);
}
public void startElement(String namespaceURI, String localName,
  String qName, Attributes atts) throws SAXException {









}
```

```
  public void endElement(String namespaceURI, String localName, String qName) throws SAXException {
  }
  public void endDocument() throws SAXException {
  }

  public static void main(String[] args) throws Exception {
    InputSource source = new InputSource(new FileInputStream("auction.xml"));
    XMLReader xr = XMLReaderFactory.createXMLReader();
    RunSAX rs = new RunSAX();
    xr.setContentHandler(rs);
    xr.parse(source);
  }
}
```

Aufgabe 6: (8)

Betrachten Sie folgende XQuery **auctions.xq**:

```
<actn>
{
  for $p in doc("auctions.xml")//product
  order by $p/@auctionEnd descending
  let $uu := doc("auctions.xml")//user/@username/string()
  return <p>
  {$p/productname}
  <info>
  {
    for $u in $uu
    let $f := doc("auctions")//bidproduct[@id=$p/@id]/bid[@user=$u]
    where $u != "Bart"

    return if(count($f) = 0) then
      <hb w="{$u}">0</hb>
    else
      <hb w="{$u}">1</hb>
  }
  </info></p>
}
</actn>
```

Geben Sie nun die Ausgabe von **auctions.xq** angewandt auf **auctions.xml** an.

Sie müssen sich nicht um Whitespaces kümmern.

Aufgabe 7: (8)

Betrachten Sie das folgende XSLT Dokument **auctions.xsl**:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
<xsl:output method="text"/>

 <xsl:template match="auctions">
   <xsl:apply-templates select="//bidproduct"/>
 </xsl:template>

 <xsl:template match="bidproduct">
   <xsl:value-of select="//product[@id=current()/@id]/productname"/>
   ( <xsl:value-of select="//category[@id = //product[@id=current()/@id]/category/@ref][1]" /> )
   <xsl:for-each select ="bid">
     <xsl:sort select="current()" order="descending"/>
     <xsl:apply-templates select="."/>
   </xsl:for-each>
 </xsl:template>

 <xsl:template match="bid">
   <xsl:variable name="userid" select="current()/@user" />
   $ <xsl:value-of select="."/> by <xsl:apply-templates select="//user[@username = $userid]"/>
 </xsl:template>

 <xsl:template match="user">
   <xsl:choose>
     <xsl:when test="fullname">
       <xsl:value-of select="fullname"/>
     </xsl:when>
     <xsl:otherwise>
       <xsl:value-of select="lastname"/> <xsl:value-of select="firstname"/>
     </xsl:otherwise>
   </xsl:choose>
 </xsl:template>

</xsl:stylesheet>
```

Geben Sie nun die Ausgabe von **auctions.xsl** angewandt auf **auctions.xml** an.

Sie müssen sich nicht um Whitespaces kümmern.

Total points: 75

Datei **auctions.xml**: **Sie können diese Seite abtrennen!**

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE auctions SYSTEM "auctions.dtd">
<auctions>
  <products>
     <product id="p1" auctionEnd="2019-07-10">
        <productname>Web Engineering</productname>
        <description>This is the description of <a href="about:blank">Web Engineering</a></description>
        <category ref="cat1"/>
     </product>
     <product id="p2" auctionEnd="2019-06-25">
        <productname>The Great Gatsby (Movie + Soundtrack)</productname>
        <category ref="cat2"/>
        <category ref="cat3"/>
     </product>
     <product id="p3" auctionEnd="2016-06-30">
        <productname>XML in a Nutshell</productname>
        <description>XML in a Nutshell is a great book about XML.</description>
        <category ref="cat1"/>
     </product>
  </products>
  <categories>
      <category id="cat1">book</category>
      <category id="cat2">movie</category>
      <category id="cat3">music</category>
  </categories>
  <users>
    <user username="Bart"><fullname>Bart Simpson</fullname></user>
    <user username="Lisa"><firstname>Lisa</firstname><lastname>Simpson</lastname>
    </user>
    <user username="Homer"><fullname>Homer Simpson</fullname></user>
  </users>
  <bids>
    <bidproduct id="p1">
       <bid user="Bart">400</bid>
       <bid user="Lisa">410</bid>
       <bid user="Bart">450</bid>
    </bidproduct>
    <bidproduct id="p2">
       <bid user="Homer">200</bid>
       <bid user="Bart">300</bid>
    </bidproduct>
    <bidproduct id="p3">
       <bid user="Bart">150</bid>
    </bidproduct>
  </bids>
</auctions>
```