EXAM IN "SEMI-STRUCTURED DATA" **184.705**			26. 06. 2018
Study Code	Student Id	Family Name	First Name

Working time: 100 minutes.
Exercises have to be solved on this exam sheet; Additional slips of paper will not be graded.
First, please fill in your name, study code and student number. Please, prepare your student id.

Exercise 1: (12)

Consider the following XML schema file **test.xsd**:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="A">
        <xsd:complexType mixed="true" >
            <xsd:all>
                <xsd:element name="B" minOccurs="0" type="xsd:boolean"/>
                <xsd:element name="C" type="typeC"/>
            </xsd:all>
        </xsd:complexType>
        <xsd:unique name="key1">
            <xsd:selector xpath=".//C"/>
            <xsd:field xpath="@id"/>
         </xsd:unique>
    </xsd:element>

    <xsd:complexType name="typeC" mixed="false">
         <xsd:sequence>
                <xsd:element name="C" type="typeC" minOccurs="0" maxOccurs="2"/>
          </xsd:sequence>
          <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
 </xsd:schema>
```

Furthermore, consider the eight different XML files, which are listed below.

You may assume that each of the following XML files is well-formed. The point is to determine the validity according to **test.xsd**.

Check which of the following XML files are valid according to **test.xsd**.

1. `<A><C/></A>` valid ◯ invalid ◯
2. `<A>SSD<C></C></A>` valid ◯ invalid ◯
3. `<A><B>0</B></A>` valid ◯ invalid ◯
4. `<C id="42"/>` valid ◯ invalid ◯
5. `<A>SSD<B>wahr</B><C></C></A>` valid ◯ invalid ◯
6. `<A>SSD<C></C><B>1</B></A>` valid ◯ invalid ◯
7. `<A><C><C id="12"></C></C><B>0</B></A>` valid ◯ invalid ◯
8. `<A><C id="12"><C><C id="12"></C></C></C><B>0</B></A>` valid ◯ invalid ◯

(For every correct answer 1.5 points, **for every incorrect answer -1.5 points**, for every unanswered question 0 points, you can have at least 0 points on this exercise)

Exercise 2: (15)

Give concise answers to the following questions (1.5 points per correct answer).

1. Name the data model for semi-structured data?

2. Name one of the formalismus (formats) to store semi-structured data:

3. What does the Abbreviation DTD stand for?

4. How do the memory requirements of event-based parser like SAX scale with the size of the parsed XML document?

5. Which of the APIs discussed in the lecture allows random access to the entire XML document?

6. Which of the languages covered in the lecture contain XPath?

7. In the lecture we covered different languages to define schemas. Which one has the greatest expressiveness?

8. Name one of the purposes of XML Namespaces?

9. Why can URIs not be used directly as a prefix for XML elements/attributes?

10. What does the default template in XSLT do for attributes? Was macht das Default-Template in XSLT für Attribute?

The following Exercises 3 – 7 are referring to the XML document travelAgency.xml, which can be found on the last page of this exam.

Exercise 3: (12)

Complete the DTD **travelAgency.dtd**, so that XML documents structured like **travelAgency.xml** (see attachment) are valid according to this DTD. Consider the following points when creating the DTD:

- The `sportcamps` element contains a arbitrary number of `summercamp` and `wintercamp` elements.
- The `customers` element contains a arbitrary number of `customer` elements.
- The `bookings` element contains a arbitrary number of `booking` elements.
- `summercamp` and `wintercamp` Elements contain one `name` element, maybe a `venue` element, at least one `sport` element, one `duration` element und a arbitrary number of `date` elements (in the given order).
- `summercamp`, `wintercamp` and `customer` elements have a attribute that is a unique ID.
- In `venue` elements the country might be marked by `country` element. Each `venue` element has an attribute named `id`.
- `booking` elements contain a `customerid` attribute that refers to the ID of a customer, and a attribute `campid` that refers to the ID of a camp. Moreover, there is an optional `date` attribute.
- If not specified make reasonable assumptions on the types of elements.

File **travelAgency.dtd**:

```
<!ELEMENT travelAgency (sportcamps,customers,bookings)>
```

Exercise 4: (10)

Consider the following XPath queries applied to the document **travelAgency.xml** (see attachement).

- If the given XPath expression selects the empty node set, write as output "empty output"
- If a number is selected as the result (count,sum,...), write as output this number.

Now give the outputs of the respective XPath queries:

```
sum(//summercamp/duration)
```

```
//customer[@id=//booking/@customerid][2]
```

```
//sportcamps/*[@id=//booking/@campid][last()-1]/name/text()
```

```
//customer[last()][@id=//booking/@customerid]
```

```
//summercamp[not(@id=//booking/@campid)][last()]/name
```

Exercise 5: (8)

Consider the following XQuery expression **xquery.xq**:

```
<venues>
{
   for $vid in distinct-values(//venue/@id)
   let $v := distinct-values(//venue[@id = $vid])
   where count(//sport[preceding-sibling::venue/@id = $vid])>2
   order by $v
   return
       <venue><name>{$v}</name>
       {
          for $s in //sport[preceding-sibling::venue/@id = $vid]
          order by $s
          return
              <sport>{$s/text()}</sport>
       }
       </venue>
}
</venues>
```

Now give the output of **xquery.xq** applied to **travelAgency.xml**.

You do not need to consider whitespace issues.

Exercise 6: (10)

Create an XSLT-Stylesheet **travelAgency.xsl**, which returns, applied to documents of the form **travelAgency.xml** the following output:

> The output is an HTML document with participant lists for all winter camps.
>
> For each winter camp, an element `h1` with the name of the camp is created as a heading.
>
> In a `ul` element, the camp participants are listed. For each camp participant a `li` element is created. The content of the `li` element is the name of the camp participant. Camp participants can be determined via the `bookings` subtree.
>
> In addition, the number of camp participants in a `p` element should be output.

Consider the following output that your XSLT-Stylesheet **travelAgency.xsl** shall return applied to **travelAgency.xml**:

```
<h1>Advanced Skiing</h1>
<ul>
    <li>Arthur Dent</li>
</ul>
<p>Total participants: 1</p>
```

Note: You only need to output the contents of the `body` element.

Now write the XSLT-Stylesheet **travelAgency.xsl**.

Exercise 7: (8)

Consider the following Java class and specify the output of the class when using the file **travelAgency.xml** as input.

```
public class RunSAX extends DefaultHandler {
    private String eleText, id; private int total = 0;
    private HashMap<String, List<String>> s = new HashMap<String, List<String>>();
    private HashMap<String, List<String>> c = new HashMap<String, List<String>>();

    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 {
        if ("summercamp".equals(localName) || "wintercamp".equals(localName)) {
            id = atts.getValue("id");
            s.put(id, new LinkedList<String>());
        }

        if ("booking".equals(localName)) {
            id = atts.getValue("customerid");
            if (!c.containsKey(id))
                c.put(id, new LinkedList<String>());
            c.get(id).add(atts.getValue("campid"));
        }
    }

    public void endElement(String namespaceURI, String localName, String qName) throws SAXException {
        if ("sport".equals(localName)) s.get(id).add(eleText);
    }

    public void endDocument() throws SAXException {
        for (String key : c.keySet()) {
            System.out.print(key + ": ");
            String out = "";
            for (String cid : c.get(key))
                for (String sp : s.get(cid))
                    out += sp + ", ";
            System.out.println(out.substring(0,out.length()-2));
        }
    }

    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("Usage: java RunSAX <input.xml>");
            System.exit(1);
        }

        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);
    }
}
```

Give the output here:

Total points: 75

File **travelAgency.xml**: **You may separate this page!**

```
<travelAgency>
    <sportcamps>
        <summercamp id="c1">
            <name>Fun Diving</name>
            <venue id="v1">Krk, <country>Croatia</country></venue>
            <sport>Scuba Diving</sport>
            <duration>7</duration>
            <date>5.7.2018</date>
            <date>5.8.2018</date>
        </summercamp>
        <summercamp id="c2">
            <name>Hiking &amp; Biking with kids</name>
            <venue id="v2">Hinterglemm, Austria</venue>
            <sport>Mountain Biking</sport>
            <sport>Hiking</sport>
            <duration>6</duration>
            <date>15.7.2018</date>
        </summercamp>
        <summercamp id="c3">
            <name>Big Waves Tour</name>
            <venue id="v3">Honolulu, Hawaii</venue>
            <sport>Surfing</sport>
            <sport>Sailing</sport>
            <duration>14</duration>
            <date>1.8.2018</date>
        </summercamp>
        <wintercamp id="c4">
            <name>Advanced Skiing</name>
            <venue id="v2">Hinterglemm, Austria</venue>
            <sport>Skiing</sport>
            <sport>Snow Board</sport>
            <duration>7</duration>
        </wintercamp>
        <summercamp id="c5">
            <name>Sailing for beginners</name>
            <venue id="v4">Rust, <country>Austria</country></venue>
            <sport>Sailing</sport>
            <duration>5</duration>
            <date>15.8.2018</date>
        </summercamp>
    </sportcamps>
    <customers>
        <customer id="p1">Sophie Haas</customer>
        <customer id="p2">Samuel Mumm</customer>
        <customer id="p3">Arthur Dent</customer>
        <customer id="p4">Harvey Dent</customer>
        <customer id="p5">Miles O'Brien </customer>
    </customers>
    <bookings>
        <booking customerid="p2" campid="c2" date="15.7.2018"/>
        <booking customerid="p3" campid="c4" date="6.1.2017"/>
        <booking customerid="p1" campid="c1" date="5.7.2018"/>
        <booking customerid="p1" campid="c3" date="5.8.2016"/>
        <booking customerid="p4" campid="c5" />
    </bookings>
</travelAgency>
```