EXAM IN "SEMI-STRUCTURED DATA" **184.705**			12. 01. 2016
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 DTD schema file **test.dtd**:

```
<!ELEMENT A (B|(C,((D+,B)|B?)))>

<!ELEMENT B (#PCDATA)>

<!ELEMENT C (#PCDATA)>

<!ELEMENT D (#PCDATA)>
```

Consider additionally the following eight different XML files. All of the following files are well-formed. In this exercise you have to decide, which of the following are valid according to **test.dtd**.

1. `<A/>` valid ◯ invalid ⊗
2. `<B>xyz</B>` valid ⊗ invalid ◯
3. `<B><C>xyz</C></B>` valid ◯ invalid ⊗
4. `<A><C>xyz</C></A>` valid ⊗ invalid ◯
5. `<A><D>xyz</D></A>` valid ◯ invalid ⊗
6. `<A><C>xyz</C><D></D><D>xyz</D><D></D><B>xyz</B></A>` valid ⊗ invalid ◯
7. `<A><C></C><D></D><B></B><B></B></A>` valid ◯ invalid ⊗
8. `<A><B><B>xyz</B></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)
Decide which of the following statements is true or false.

1. In semi-structured data entities in the same class have the same attributes. true ◯ false ⊗
2. An XML document can be seen as a graph. true ⊗ false ◯
3. XML is a W3C standard. true ⊗ false ◯
4. The "X" in XML stands for Extensible. true ⊗ false ◯
5. DTDs can be used for querying XML documents. true ◯ false ⊗
6. An XML Schema is not an XML document. true ◯ false ⊗
7. DOM is a tree-based API for manipulating XML documents. true ⊗ false ◯
8. XPath is a language for extracting parts of an XML document. true ⊗ false ◯
9. Every XPath expression is an XQuery expression. true ⊗ false ◯
10. XSLT is more powerful than XQuery. true ◯ false ⊗

(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)

The following Exercises 3 – 7 are referring to the XML document groups.xml, Exercises 6 is referring to the XML document groups-xsl.xml. Both can be found on the last page of this exam.

Exercise 3: (14)

Create an XML Schema document **groups.xsd** such that the **groups.xml** document is valid. Consider the following specification:

- The root element of the document is called `groups`. It contains zero or an unbounded number of `group` elements.
- The `group` element has an attribute `name` and contains either
  - zero or an unbounded number of `group` elements, or
  - zero or an unbounded number of `user` or `userref` elements in arbitrary order.
- The `user` element has mixed content, a `name` attribute and may contain a `description` element.
- The `userref` element is an empty element and has an attribute `name`.
- The `name` element contains a string.
- Add a key `userKeys` identifying all `user` elements by their `name` attribute.
- Add a key reference `userKeyRef`. The `name` attribute of the `userref` elements refer to the `userKeys`.

File **groups.xsd**:

```
<!-- More space on the following page! -->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <xsd:element name="groups" type="groupsType">
       <xsd:key name="userKeys">
           <xsd:selector xpath=".//user"/>
           <xsd:field xpath="@name"/>
       </xsd:key>

       <xsd:keyref name="userRefs" refer="userKeys">
           <xsd:selector xpath=".//userref"/>
           <xsd:field xpath="@name"/>
       </xsd:keyref>
   </xsd:element>

   <xsd:complexType name="groupsType">
       <xsd:sequence>
           <xsd:element name="group" type="groupType" minOccurs="0" maxOccurs="unbounded"/>
       </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="groupType">
       <xsd:choice>
          <xsd:element name="group" type="groupType" maxOccurs="unbounded" />
          <xsd:choice maxOccurs="unbounded">
             <xsd:element name="user" type="userType" />
             <xsd:element name="userref" type="userrefType" />
          </xsd:choice>
       </xsd:choice>
       <xsd:attribute name="name" type="xsd:string" use="required"/>
   </xsd:complexType>
```

```
       <xsd:complexType name="userType" mixed="true">
       <xsd:sequence>
          <xsd:element name="description" type="xsd:string" minOccurs="0" />
       </xsd:sequence>
       <xsd:attribute name="name" type="xsd:string" use="required"/>
    </xsd:complexType>


    <xsd:complexType name="userrefType">
        <xsd:attribute name="name" type="xsd:string" use="required"/>
    </xsd:complexType>

</xsd:schema>
```

Exercise 4: (8)

Consider the following XPath expressions and evaluate them over the **groups.xml** document.

- If the expression selects several nodes, separate the output with whitespaces.
- If the XPath expression selects no nodes, write "No output!".

Write the output of the following expressions:

```
count(//group[@name="DBAI"]/*)
```

2

```
//group[@name="DBAI"]//group/@name
```

SPARQL FPT core

```
count(//group[@name="DBAI"]/following::*)
```

4

```
//user[@name=//userref/@name]/description
```

No output!

Exercise 5: (6)

Consider the following XQuery statement **groups.xq**:

```
for $g in //group, $u in $g//user
where $g/@name="users"
order by $u/@name
return <group name="{$g/@name}">{string($u/@name)}</group>
```

Write the output of **groups.xq** evaluated over **groups.xml** here.
Whitespaces don't have to be formatted correctly.

```
<group name="users">martin</group>
<group name="users">thomas</group>
<group name="users">wolfgang</group>
```

Exercise 6: (8)

Create an XSLT stylesheet **groups.xsl** that, after applied to **groups.xml**, outputs the XML document **groups-xsl.xml**. The idea is to generate a document which lists all groups and the usernames occurring in the subtree of every group. This means:

- The root element is `groups`.
- For each `group` element: Create an element which has as name the content of the `name` attribute of the `group`. Apply templates to all `user` and `userref` descendants.
- For each `user` element: Create a `user` element, that copies the content of the `name` attribute into its `name` attribute.
- For each `userref` element: Create a `user` element, that copies the content of the `name` attribute into its `name` attribute.

Write the stylesheet here **groups.xsl**.

File **groups.xsl**:

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/>

   <xsl:template match="groups">
      <groups><xsl:apply-templates select="//group" /></groups>
   </xsl:template>

   <xsl:template match="group">
      <xsl:element name="{@name}">
         <xsl:apply-templates select=".//user" />
         <xsl:apply-templates select=".//userref" />
      </xsl:element>
   </xsl:template>

   <xsl:template match="user">
      <user name="{@name}" />
   </xsl:template>

   <xsl:template match="userref">
      <user name="{@name}" />
   </xsl:template>

</xsl:stylesheet>
```

Exercise 7: (12)

Complete the method `addUser` in the Java class `InsertUser`. Use DOM to insert a new user into the XML document **groups.xml** loaded in the variable `doc`. For this, proceed as follows:

(1) Use `xPath` to find the group with the name `groupName`.

(2) Use the method `findDeepestGroupChild` to find the `group` element which has no other `group` element as child.

(3) Create a new element `user` with `userName` in its `name` attribute. Add this element to the node found in Step (2).

You don't need to output the XML document `doc` and you don't need to handle any exceptions.

```
public class InsertUser  {
    private static XPath xPath = XPathFactory.newInstance().newXPath();
    Document doc;

    public InsertUser(Document doc) {
        this.doc = doc;
    }

    //This method finds the group element with no group element as child
    //in the subtree rooted at the Node n
    private Node findDeepestGroupChild (Node n) {
        if (n.hasChildNodes()) {
            NodeList children = n.getChildNodes();
            for ( int i = 0; i<children.getLength(); i++ )
                if (children.item(i).getNodeName().equals("group"))
                    return findDeepestGroupChild(children.item(i));
            return n;
        } else
            return n;
    }

    public void addUser (String groupName, String userName) throws Exception {

        //Find the group element with name groupName
        XPathExpression xpe = xPath.compile("//group[@name = \"" + groupName + "\"]");
        NodeList groupList = (NodeList) xpe.evaluate(doc, XPathConstants.NODESET);

        //Traverse to the deepest group element
        Node group = findDeepestGroupChild(groupList.item(0));

        //Insert a new user element into group
        Element userEle = doc.createElement("user");
        userEle.setAttribute("name", userName);
        group.appendChild(userEle);
    }

}
```

Total points: 75

You can remove this sheet!

File **groups.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<groups>
   <group name="users">
      <group name="DBAI">
         <group name="SPARQL">
            <user name="wolfgang">
                This is the user
                <description>Wolfgang</description>.
            </user>
         </group>
         <group name="FPT">
            <group name="core">
               <user name="martin" />
            </group>
         </group>
      </group>
      <group name="KBS">
         <userref name="martin" />
         <user name="thomas">
            This is the user
            <description>Thomas</description>.
         </user>
      </group>
   </group>
</groups>
```

File **products-xsl.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<groups>
   <users>
      <user name="wolfgang"/>
      <user name="martin"/>
      <user name="thomas"/>
      <user name="martin"/>
   </users>
   <DBAI>
      <user name="wolfgang"/>
      <user name="martin"/>
   </DBAI>
   <SPARQL>
      <user name="wolfgang"/>
   </SPARQL>
   <FPT>
      <user name="martin"/>
   </FPT>
   <core>
      <user name="martin"/>
   </core>
   <KBS>
      <user name="thomas"/>
      <user name="martin"/>
   </KBS>
</groups>
```