	Project #1 Integração de Sistemas/ Enterprise Application Integration
UNIVERSIDADE DE COIMBRA FACULDADE DE CIÊNCIAS E TECNOLOGIA *Departamento de Engenharia Informática*	2017/18 – 1ˢᵗ Semester MEI **Deadline: 2017-10-13**

Notas:

A fraude denota uma grave falta de ética e constitui um comportamento não admissível num estudante do ensino superior e futuro profissional. Qualquer tentativa de fraude pode levar à reprovação na disciplina tanto do facilitador como do prevaricador.

Os trabalhos entregues serão verificados por software de deteção de cópias.

XML and XML Manipulation, Java Message Service (JMS) and Message Oriented Middleware (MOM)

Objectives

- Learn **XML technologies**. You will learn XML, XSD, XSL, XSLT and XPATH. This project is mostly about XML processing.

- Learn how to create simple asynchronous and message-oriented applications.

Final Delivery

- This assignment contains two parts: one is for training only, and does not count for the evaluation. You should only deliver the other part.
- **You are not allowed to use JMS 1.1. You must use JMS 2.0.**
- You must submit your project in a zip file with the **source** of a complete Maven project, using Inforestudante. Do not forget to associate your work colleague during the submission process.
- The Maven project(s) should produce executable .jar file(s)
- The submission contents are:
  - o Source code of the requested applications ready to compile and execute.
  - o A small report in pdf format (5 pages max) about the implementation of the project.

Resources

Jsoup
- Jsoup Java HTML Parser, with best of DOM, CSS, and jquery: http://jsoup.org
- Manual at: http://jsoup.org/cookbook/

XML, XSD, XSL and XSLT
- XML: http://www.w3schools.com/xml
- XSD: http://www.w3schools.com/xml/xml_schema.asp
- XPATH: https://www.w3schools.com/xml/xpath_intro.asp
- "Chapter 2: Understanding XML", in J2EE 1.4 Tutorial http://download.oracle.com/javaee/1.4/tutorial/doc/
- JAXB Tutorial – Java.net: http://jaxb.java.net/tutorial/index.html
- Trang – http://www.thaiopensource.com/relaxng/trang.html

Processing XML/XSLT in Java
- "Chapter 7: Extensible Stylesheet Language Transformations", in J2EE 1.4 Tutorial
  (Especially, the part "How XPath Works" and "Transforming XML Data with XSLT")
  http://download.oracle.com/javaee/1.4/tutorial/doc/
- David Jacobs, "Rescuing XSLT from Niche Status – A Gentle Introduction to XSLT through HTML Templates", http://www.xfront.com/rescuing-xslt.html
- G. Ken Holman, "Getting started with XSLT and XPath", in XML.COM http://www.xml.com/pub/2000/08/holman/s2_1.html
- Paul Grosso and Norman Walsh, "XSL Concepts and Practical Use", in NWalsh.COM
  http://nwalsh.com/docs/tutorials/xsl/xsl/frames.html

Java Message Service
- http://docs.oracle.com/javaee/7/api
- To run Java Message Service, you will need to install WildFly (version 11): http://wildfly.org
- What's New in JMS 2.0, Part One: Ease of Use: http://www.oracle.com/technetwork/articles/java/jms20-1947669.html
- Java Message Service 2.0 with WildFly 8:

Advice: Skim all the links above before starting to read anything in detail.
Note: You have short examples of some of the technologies in the next section.

XML Training Part (doesn't count for evaluation)

1. Use an online tool, or install a tool like *trang* to automatically produce the XSD for the following XML. Change the XSD, to ensure that <direction> can only be one of "dgsg|boinc" or "dgsg|xtremweb", while <timestamp> must be positive. Note that you should always check if the tool inferred the correct schema, or if it requires some manual adjustment.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<report timestamp="1308046204104" timezone="GMT" version="1.1">
<metric_data>
    <metric_name>cpus_available</metric_name>
    <timestamp>1308046204003</timestamp>
    <value>0.0</value>
    <type>uint32</type>
    <units>cpus</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
</metric_data>
<metric_data>
    <metric_name>gflops</metric_name>
    <timestamp>1308046204056</timestamp>
    <value>0.0</value>
    <type>float</type>
    <units>gflops</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
</metric_data>
<metric_data>
    <metric_name>past_workunits</metric_name>
    <timestamp>1308046204058</timestamp>
    <value>0.0</value>
    <type>uint32</type>
    <units>wus</units>
    <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
    <direction>dgsg|boinc</direction>
</metric_data>
<metric_data>
    <metric_name>waiting_workunits</metric_name>
    <timestamp>1308046204059</timestamp>
    <value>0.0</value>
    <type>uint32</type>
```

```xml
      <units>wus</units>
      <spoof>EDGITest|dsp:EDGITest|dsp</spoof>
      <direction>dgsg|boinc</direction>
    </metric_data>
    <metric_data>
      <metric_name>success_rate</metric_name>
      <timestamp>1308046204061</timestamp>
      <value>1.0</value>
      <type>float</type>
      <units>percentage</units>
      <spoof>EDGITest|dsp:EDGITest|dsp</spoof>
      <direction>dgsg|boinc</direction>
    </metric_data>
    <metric_data>
      <metric_name>past_workunits_24_hours</metric_name>
      <timestamp>1308046204064</timestamp>
      <value>0.0</value>
      <type>uint32</type>
      <units>wus</units>
      <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
      <direction>dgsg|boinc</direction>
    </metric_data>
    <metric_data>
      <metric_name>cpus_available</metric_name>
      <timestamp>1308046204066</timestamp>
      <value>0.0</value>
      <type>uint32</type>
      <units>cpus</units>
      <spoof>EDGITest|dsp:EDGITest|dsp</spoof>
      <direction>dgsg|boinc</direction>
    </metric_data>
    <metric_data>
      <metric_name>success_rate</metric_name>
      <timestamp>1308046204067</timestamp>
      <value>1.0</value>
      <type>float</type>
      <units>percentage</units>
      <spoof>EDGITest|fusion:EDGITest|fusion</spoof>
      <direction>dgsg|boinc</direction>
    </metric_data>
    <metric_data>
      <metric_name>gflops</metric_name>
      <timestamp>1308046204092</timestamp>
      <value>0.0</value>
      <type>float</type>
      <units>gflops</units>
      <spoof>EDGITest|dsp:EDGITest|dsp</spoof>
      <direction>dgsg|boinc</direction>
    </metric_data>
</report>
```

2. Now use the XML Binding Compiler (*xjc*) command-line tool to generate Java classes that represent the XML Schema that you generated. After this, write a simple program that performs two functions:
    a) Unmarshalls the information contained in the example XML to Java objects (the generated classes will hold the information);
    b) Marshalls the same information, now contained in Java Objects, back to XML.

3. Let's now try the Java-first approach. In this case, you will be writing the Java classes yourself, and using the JAXB notation (e.g., annotations). You should use JAXB to output the following XML:

a)
```xml
<?xml version="1.0" encoding="UTF-8"?>
<class>
        <student>
                <name>Alberto</name>
                <age>21</age>
        </student>
        <student>
                <name>Patricia</name>
                <age>22</age>
        </student>
    <student>
                <name>Luis</name>
                <age>21</age>
        </student>
</class>
```

b)
```xml
<?xml version="1.0" encoding="UTF-8"?>
<class>
        <student id="201134441110">
                <name>Alberto</name>
                <age>21</age>
        </student>
        <student id="201134441116">
                <name>Patricia</name>
                <age>22</age>
        </student>
    <student id="201134441210">
                <name>Luis</name>
                <age>21</age>
        </student>
</class>
```

c) (An XML equivalent to this is also acceptable)
```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated automatically. Don't change it. -->
<class xmlns="http://www.dei.uc.pt/EAI">
  <student xmlns="" id="201134441110">
     <name>Alberto</name>
```

```
     <age>21</age>
   </student>
   <student xmlns="" id="201134441116">
     <name>Patricia</name>
     <age>21</age>
   </student>
   <student xmlns="" id="201134441210">
     <name>Luis</name>
     <age>21</age>
   </student>
 </class>
```

d)
```
 <?xml version="1.0" encoding="UTF-8"?>
 <?xml-stylesheet type="text/xsl" href="test.xsl"?>
 <!-- Generated automatically. Don't change it. -->
 <h:class xmlns:h="http://www.dei.uc.pt/EAI">
   <h:student id="201134441110">
     <name>Alberto</name>
     <age>21</age>
   </h:student>
   <h:student id="201134441116">
     <name>Patricia</name>
     <age>21</age>
   </h:student>
   <h:student id="201134441210">
     <name>Luis</name>
     <age>21</age>
   </h:student>
 </h:class>
```

e)
```
 <?xml version="1.0" encoding="UTF-8"?>
 <class>
     <student id="201134441110">Alberto</student>
     <student id="201134441116">Particia</student>
     <student id="201134441210">Luis</student>
 </class>
```

4. Try to manually write the XML Schema Definition for the XML of exercise 3.a).

5. Write an XSL file capable of outputting the XML data of Exercise 2 into an HTML table. Use a web browser to apply and visualize the transformation (you could also use a Java library, such as Xalan, for this purpose).

JMS Training Part (doesn't count for evaluation)

1. Run the example available at:

http://eai-course.blogspot.pt/2015/03/java-message-service-20-with-wildfly-8.html

2. Assume now that the sender needs to receive a reply, but you do not want to configure a dedicated queue for that. Which mechanism could you use? Write the necessary code, sending a reply with a set of key-values. (HINT for the sender: use the createTemporaryQueue() and a TextMessage in the JMSContext; then, use the setJMSReplyTo() in the message).

3. Write code that sends text messages to multiple subscribers at once.

4. In the previous code, what happens to the messages that arrive at the topic, before the subscriber makes the subscription? Assume now, that a client subscribes a topic, leaves and then subscribes again. We want to know what happens to the messages that enter the topic, while this client is out. Will it receive the messages or not? To ensure that the client receives these messages, which changes do you need to do? Write a new client with these properties and try the code to see if it works. You should check this message: http://eai-course.blogspot.pt/2012/09/a-few-variations-over-jms.html[1].

5. How do queues behave when there is no receiver? Do they keep the messages or do they drop the messages?

6. Consider now that you only want to receive messages concerning the Enterprise Application Integration course. How can you avoid the remaining? Will this work, both for queues and for topics? Implement a working example.

7. Assume now that you need to send an XML message for a topic. Which kind of JMS messages should you use? (You do not need to implement the response)

8. Explain the parameter of the method `JMSContext.createContext()`. What are the different types of acknowledgment available and what are their differences?

9. Explain the difference between persistent messages and durable subscriptions.

---

[1] This message is deprecated because it refers to Java Message Service 1.1. However, you will find pretty much the same functionalities in the `JMSContext` interface implementation.
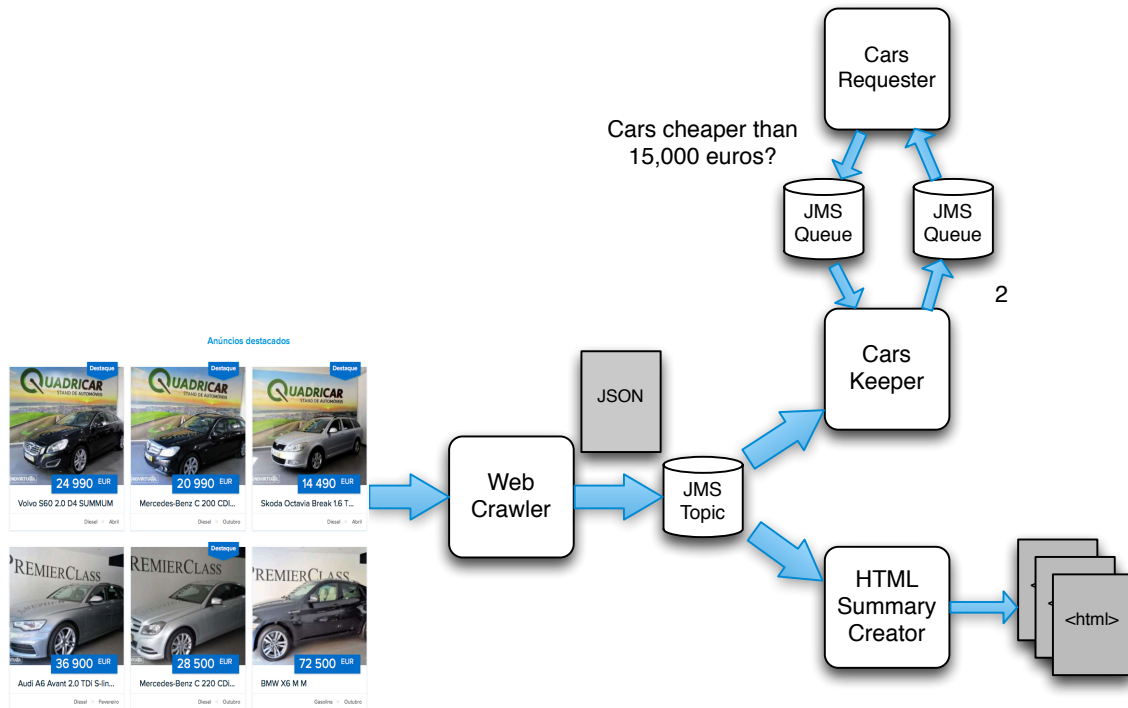
Project Part (for evaluation)

Figure 1 – The information flow

In this assignment, you should create four applications. The first one is a *Web Crawler* that collects information about a web site of cars[2], extracts the relevant data to XML, and sends it to a Java Message Service Topic. This topic serves two other applications that process the data. One of the applications, *Cars Keeper*, keeps information about the collected cars. The *Cars Requester* asks for information from the *Cars Keeper*. The other application, *HTML Summary Creator*, summarizes the information and creates HTML files for later visualization. Figure 1 illustrates this scenario. The four applications are described in the following paragraphs.

The Web Crawler

The *Web Crawler* is a stand-alone command-line application that reads a web page and sends an XML message to a JMS Topic, carrying details of featured cars from the site (https://www.standvirtual.com/destaques/). You should use an HTML parser (e.g., Jsoup), to get the data from the all the featured cars, which should be close to 100 in 3 different web pages. **You should not parse the web page directly using regular expressions.** Nevertheless, you can use regular expressions to extract small pieces of data from the results of the HTML parser. For example, you might find a string that looks like "1500 cm3" and use regular expressions to transform the string as you wish.

---

[2] https://www.standvirtual.com/.

Once you get the DOM of the web page, you will need to convert it to XML. If this helps, you can do as follows, although creating the Java classes from scratch is not only possible, but simple:

- Define the XML schema (this may involve the *trang* tool or a web site, to create XSD from an XML sample). **You must include an XML schema file (XSD) as part of your final submission and be ready to explain it;**
- From the XML schema, generate the Java classes using the XML binding compiler, *xjc*);
- Once you have the Java classes that can keep the data, you can instantiate and use them in the normal way, in the *Web Crawler* (or other application) source code.

Each time the *Crawler* runs, it parses the web page, creates and populates the Java objects that keep the web site's data, and sends a JMS message with the XML document to the JMS Topic. If the topic is down for some reason, the Crawler should keep a file with the contents it was going to publish. The next time it runs, it will try to send the contents of the file (and delete it, if successful), instead of crawling the web.

You are responsible for defining the exact format of the XML messages. You should try to extract as much information as possible for each car. One of the important points for evaluation will be the quality of the XML and XSD of your application. You can take different options here, but you are advised to discuss them with the Professor.

HTML Summary Creator

This application should run permanently, waiting for XML messages on the JMS topic. When this happens, the *HTML Summary Creator* should translate the XML into an HTML file (you should keep one file per reading of the *Crawler*). For this, you should use an XSL template.

You are pretty much free to define the HTML as you want, but you should include all the data collected by the *Crawler*. Organization of the page is important. You may either use a web browser with a built-in XSLT engine (e.g., Firefox) to apply the transformation and display the resulting HTML page, or you might do this transformation inside the *HTML Summary Creator*.

Cars Keeper

The purpose of this application is to keep track of car information, as the *Web Crawler* sends it. Keeping the cars in memory is sufficient for the sake of this assignment. The *Cars Keeper* must listen to two different destinations: the topic where the Crawler sends its messages (a durable subscription is necessary here) and a queue where the

Cars Requester(s) asks for information. Take due to care to write this part of the application correctly.

Note: Use durable subscriptions in the *HTML Summary Creator* and *Cars Keeper*, to ensure that even if these applications fail, the Topic will keep the messages for later retrieval. You should also check the XML files received in this programs against the XML schema, to avoid consuming incorrect XML.

Cars Requester

The *Cars Requester* allows the user to introduce some reference of the car (e.g., brand or year), sends this data to the *Cars Keeper* and gets a car or list of cars back. To get the response, the *Cars Requester* must create a **temporary queue** and add a reference to this queue in the request it does to the *Keeper*. This easily allows several *Requesters* to run at the same time. The *Keeper* should match all the information it has, based on what the *Requester* provides. For example, if the *Requester* only asks for "Toyota", the *Keeper* should send data of all the Toyotas. You do not need to use complex regular expressions in these queries.

Maven

To manage your project development cycle, you should use Maven.

Grading

Grading is performed according to a grid that will be published together with this assignment and according to individual student defenses.

The project should be made in groups of **2** students. I do expect all the members of the group to be fully aware of all the parts of the code that is submitted. Work together!