UNIVERSIDADE DE COIMBRA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
Departamento de Engenharia Informática

Assignment #1
Integração de Sistemas/ Enterprise Application Integration

2021/22 – 1st Semester
MEI

Deadline: 2021-10-08

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 plágio.

Data Representation and Serialization Formats

Objectives

- Learn text and binary formats for data serialization
- Understand the differences between the most important formats
- This assignment will involve data representation formats, such as XML, JSON, Protocol Buffers, or MessagePack.

Resources

Maven:
- Students are advised to use Maven to manage their projects.
- Maven Tutorial: https://www.tutorialspoint.com/maven/

XML:
- XML: http://www.w3schools.com/xml
- JAXB Tutorial – Java.net: https://www.javatpoint.com/jaxb-tutorial

Note: starting on version 9, the Java Architecture for XML Binding (JAXB) is no longer part of the Java Standard Edition. This causes a number of problems with the use of this architecture. You may resort to Maven for the rescue: https://stackoverflow.com/questions/43574426/how-to-resolve-java-lang-noclassdeffounderror-javax-xml-bind-jaxbexception-in-j/46455026.

However, you should note that the version numbers of the dependencies are not right.

- Xalan: http://xml.apache.org/xalan-j/

JSON
- Homepage: https://developers.google.com/protocol-buffers/

Protocol Buffers
- Homepage: https://developers.google.com/protocol-buffers/
- Maven dependency: https://mvnrepository.com/artifact/com.google.protobuf/protobuf-java
- Compiler Download: https://github.com/protocolbuffers/protobuf/releases

MessagePack
- Homepage: https://msgpack.org

Java XML Training Part (doesn't count for evaluation)

Before you start, you may create a Maven project using the following command line, which initializes a "quickstart" archetype:

```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes
-DarchetypeArtifactId=maven-archetype-quickstart  -DgroupId=uc.mei.is
-DartifactId=simple-jaxb -Dversion=0.0.1
```

Also, refer to https://mkyong.com/java/jaxb-hello-world-example/ for the dependencies you need to insert in the pom.xml file and for a base example using JAXB.

1. Using JAXB, write the Java classes and the marshalling code that outputs the following XML:

a)

```
<?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 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 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>
```

2. 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 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>
```

```
      <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>
```

3. Now, create an XML file with a catalog of books and use XSLT to display the books in an HTML table. A simple way of accomplishing this task is by means of using Xalan-Java, referenced above.

Description of the Assignment (for Evaluation)

In this assignment, students will learn and compare data representation technologies. As a result, they should produce a pdf report with a comparison between one text and one binary format: XML or JSON vs. Google Protocol Buffers or MessagePack. For example, XML vs. MessagePack or XML vs Protocol Buffers, but neither XML vs. JSON, nor Protocol Buffers vs. MessagePack. Other options are possible and welcome under the professor's guidance.

Students should perform their comparison taking into consideration multiple parameters, including programming complexity, serialization size, and serialization and deserialization speed. Students should try to be as objective as possible. The report should also include a comparison of the use cases for each technology.

For fairness, students should define a common data structure that they will use for the comparisons. The basic data structure to use is a pet-owner many-to-one relationship. Each pet has the following information: a unique identifier; name; species; gender; weight; birth date; and free-form description. Each owner has a unique identifier; name; birth date; telephone; and address. Ownership is expressed by means of a foreign key in the pet. I.e., in addition to the pet's data, the pet needs an extra field with the identifier of the owner. Students are free to use **additional** data structures, if they wish to exercise specific details of the technologies, but they must not simplify this base one.

Students may wish to try very large sets of pets and owners, or many repetitions of the operations, to reduce the impact of random components in the performance times. Students should also do a reasonable number of experiments, to consider significant averages and standard deviations. Students should be careful about the places in the code where they register time. They should also try to use similar code as much as possible for the text and binary formats, to improve fairness. For the benefit of the report, students might want to measure the sizes of the data structures in memory.

There is no restriction of programming language or libraries, but students are encouraged to discuss their choices with the professor, to avoid any sort of shortcuts. Students might use different languages to serialize and deserialize data. However, they should compare the same operations using the same language. For example, if they write the XML serialization code in Java and the XML deserialization code in Python, then, ProtoBuf serialization should also be in Java, whereas the ProtoBuf deserialization should be in Python.

In the report, students should properly describe the conditions of the experiments, e.g., computer characteristics, technologies and languages used, their versions, and so on. The experiment should be repeatable from the report. Students must include some crucial pieces of their source code in the report (as attachment). For example, students should add data structures and the points of code where they measure times.

The report should include a short description of the data representation formats that students should use to support a brief critical discussion of results, e.g., why is MessagePack faster than XML.

Final Delivery

- Students should deliver their pdf[1] report at Inforestudante[2]. Students should be careful about the size they pick for the report. Useless verbosity is not rewarded.
- The report should be written by groups of **2** students. I do expect all the members of the group to be fully aware of all the parts of the code used. Work together!
- Grades will be based on the quality of the report: how do students describe the data representation formats, the experiment, presentation, and discussion of results; how careful were they while doing the experiments; and how many experiments did they run, to cover the different behaviors of the technologies.

[1] Delivering the report in a proprietary format, like Word, might involve a penalty in the grade.

[2] Students without access to Inforestudante should send an email to the professor with the assignment.

Some Common Errors

To guide students in their report, I summarize some of the most common mistakes that students performed a couple of years ago in the corresponding assignment:

- Unstructured document without section numbers.
- Failing to make the experimental settings explicit (e.g. the computer used, the trials ran).
- Not including the data structure used.
- Not stating the points of code where they compute the times.
- Not including the sizes of the serialized structure in the experiment.
- Figures without numbers.
- Too many plots that are repetitive and convey little information.
- Too little text alongside the plots.
- A lot of source code in the middle of the text.
- No analysis of the plots (this should go beyond repeating what is in the plots). A true explanation of what is in the plots should be given if possible. For example, in some cases, text-based representation was good for small data and bad for larger data. Why?