UNIVERSITÀ DI PISA

Università degli Studi di Pisa

Laurea Specialistica in Informatica

Advanced Programming

Final Term Paper

Start Date: 31/01/2016
Submission deadline: 19/02/2016 (send a single PDF file to attardi@di.unipi.it)

Rules:

The paper must be produced personally by the student, signed implicitly via his mail address.

You are allowed to discuss with others the general lines of the problems, provided that each student eventually formulates his own solution. Each student is expected to understand and to be able to explain his solution.

You are allowed to consult documentation from any source, provided that references are mentioned.

It is not considered acceptable:

- **to consult or setup an online forum, to request help of consultants in producing the paper**
- to develop code or pseudo-code with others
- to use code written by others
- to let others use someone"s code
- to show or to examine the work of other students.

Violation of these rules will result in the cancellation of the exam and a report to the Presidente del Consiglio di Corso di Studio.

For the programming exercises you can choose a programming language among C++, C# and Java.

The paper must:

1. be in a single PDF file, formatted readably (**font size ≥ 10 pt** with suitable margins, single column), of **no more than 10 numbered pages**, including code: for each extra page one point will be subtracted from the score.
2. include the student name
3. provide the solution and the code for each exercise separately, referring to the code of other exercises if necessary.
4. cite references to literature or Web pages from where information was taken.

Introduction

The project aims at developing a Web application framework, called APSP (AP Servlet Pages), for developing dynamic Web pages. An APSP page is an extension of HTML that may contain the following expressions:

`<%! declaration %>`	introduces a global Java declaration
`<%@ global statement %>`	introduces a global Java statement
`<%= expresssion %>`	introduces a Java expression
`<% code %>`	introduces Java code

An APSP page is translated into Java code in two steps. The first step translates the APSP code into XML, using tags <`declaration`>, <`expression`>, <`code`> and <`verbatim`>:

For instance, the following page `Access.apsp`:

```
<html>
```

```
<body>
<%@ import java.util.Date; %>
<%! int count = 0; %>
<% Date today = new Date(); %>
This page has been visited <%= count++ %> times.
<p>
Generated by APSP on <%= today %>.
</body>
</html>
```

would translate into this XML code:

```
<apsp>
<verbatim><![CDATA[<html>\n<body>\n]]></verbatim>
<declaration><![CDATA[int count = 0;]]></declaration>
<code><![CDATA[Date today = new Date();]]></code>
<verbatim><![CDATA[This page has been visited ]]></verbatim>
<expression><![CDATA[count++]]></expression>
<verbatim><![CDATA[times.\n<p>\nGenerated by APSP on ]]></verbatim>
<expression><![CDATA[today]]></expression>
<verbatim><![CDATA[.\n</body>\n</html>\n</verbatim>
</apsp>
```

The second step would translate the XML code into Java, producing:

```
import java.util.Date;

public class Access : public ApspServlet {
  static int count = 0;

   public void Get(PrintWriter doc) {
       doc.print("<html>\n<body>\n");
       Date today = new Date();
       doc.print("This page has been visited ");
       doc.print(count++);
       doc.print("times.\n<p>\nGenerated by APSP on ");
       doc.print(today);
       doc.print(".\n</body>\n</html>\n");
}
```

Exercise 1

Implement a translator, that reads an APSP file and produces the corresponding XML file.

Exercise 2

Write a trasformer in the XSLT language that translates the XML file into Java code, using an XSLT processor, for example Xalan.

Exercise 3

Write a minimal Web server (starting for example from the one available at http://www.cafeaulait.org/books/jnp3/examples/10/SingleFileHTTPServer.java) capable of accepting `GET` requests which behaves as follows depending on the URL requested:

- if the URL path is `name.apsp`, then it invokes the generator for producing the file `name.java`, compiles it, loads it, istantiates the class and invokes its method `Get()` with the proper argument so that the answer will be sent to the client.
- otherwise it sends to the client the content of the file on the server disk corresponding to the URL path.

Test the server on the example in the introduction.

Exercise 4

Modify the server to avoid reloading the requested class if it is already defined and the corresponding Java file has not changed since the last request. Explain the difference in the results obtained with this variant of the server.

Exercise 5

Discuss the various types of polymorphism and in particular compare the techiques used for implementing parametric polymorphism in C++, Java and C#. In which of these languages polymorphic methods can be virtual?