Advanced Programming

Final Term Paper

Start Date: 20/05/2014
Submission deadline: 9/06/2014 (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. **Do not include in an exercise code only needed for a later exercise**.
4. cite references to literature or Web pages from where information was taken.

Introduction

We will develop a simple Template Engine. A template is built from a string which may contain the following notations:

`{{ par }}` is an expression that refers to a template parameter of name `par`

`{{ par.attr }}` is an expression refers to attribute `attr` of parameter `par`

`{% if test %} body {% endif %}` is a statement that includes `body` if test is true

`{% for it in par %} body {% endfor %}` is a statement that repeats `body` for all values in the collection `par`. Within `body` the parameter `it` will refer to the current value of the iterator on the collection.

For example, using the following template:

```
<table>
    {% for it in popes %}
    <tr><td>{{ it.first }}</td><td>{{ it.last }}</td></tr>
    {% endfor %}
</table>
```

a program like this will generate an HTML table:

```
public class Pope {
    public Pope(string first, string last) {
          this.first = first;
          this.last = last;
}
  public string first;
  public string last;
}

Pope[] list = new Pope[] {
   new Pope("Paolo", "Sesto"),
   new Pope("Francesco", "Primo")
};
Template tableGen = Template.parse("<table> … </table>");
Environment env = new Environment();
env.bind("popes", list);
String table = tableGen.render(env);
```

The value assigned to table will be:

```
<table>
   <tr><td>Paolo</td><td>Sesto</td></tr>
   <tr><td>Francesco</td><td>Primo</td></tr>
</table>
```

Exercise 1

Design a hierarchy of classes to represent various kinds of template expressions.

Exercise 2

Implement a recursive descent parser for converting a string into a `Template` object. You can assume that the string containing the template to be expanded does not contain any occurrences of the delimiters "`{{`", "`}}`", "`{%`" and "`%}`", except for denoting template expressions or statements.

Exercise 3

Implement the class `Environment` and a polymorphic method `render()`.

Exercise 4

Add the possibility of defining parametric templates, as in this example with two parameters:

`{% lambda(x, y) %}body{% endlambda %}` defines a template with parameters `x` and `y` with body `body`

`{{ templ(x, y) }}` includes the expansion of template `templ` with parameters `x` and `y`

For example, with the following parametric template:

```
{% lambda(user) %}
    <tr><td>{{ user.first }}</td><td>{{ user.last }}</td></tr>
{% endlambda %}
```

the previous example could be written as:

```
Template tableGen =
     Template.parse("<table>{% for it in popes %}{{ row(it)
}}</table>");
Template rowGen = Template.parse("{% lambda(user) %} … {% endlambda
%}");
Environment env = new Environment();
env.bind("popes", list);
env.bind("row", rowGen);
string table = tableGen.render(env);
```

The implementation of the extension should exploit the polymorphism of methods in class `Template` and `Environment`.

Exercise 5

Provide an example of a meaningful use of construct `{% if %}` and of nested template statements and show the output produced by render.

Exercise 6

Illustrate the typical technique for implementing parametric polymorphism. In which of the languages C++, Java and C#, polymorphic methods can be virtual?