**Assignment Brief:**

**Portfolio** Part 1 – Text Template Processing Library

Using only the standard Java APIs, design and implement a Java library (consisting of one or more classes), one class of which must have a no-argument constructor, and implement the following interface:

package uk.ac.uos.i2p.assignment;

import java.util.Map;

public interface TemplateProcessor {

void loadTemplate(String name, String template);

void loadTemplates(Map<String, String> templates);

String expandTemplate(String templateName, Map<String, Object> context);

}

The loadTemplate method should store a template in the template processor with the specified name, containing the specified template text.

The loadTemplates method should store a template in the template processor with the specified name, containing the specified template text for each entry in the map.

The expandTemplate method should look through the named template, find placeholders of the form ${expression} and replace them with the value of the specified *expression*. When all placeholders have been replaced, the method should return the expanded template.

Expressions will at minimum have the following form:

|  |  |
| --- | --- |
| Expression | Result |
| *name* | Return the String value of key *name* from the context, (or “null” if it is null) |

This form of placeholder might be used in the following manner:

|  |
| --- |
| Template *“offer.tpl”* |
| Dear ${name}, We are happy to offer you a bonus of £${amount}. |

|  |
| --- |
| Context *“source/Customer123.txt”* |
| $template: offer  name: Arthur  amount: 20 |

|  |
| --- |
| Output *“destination/Customer123.txt”* |
| Dear Arthur, We are happy to offer you a bonus of £20. |

More confident students may also wish to implement the following placeholders.

|  |  |
| --- | --- |
| Expression | Result |
| (*name?expression*) | If value of key *name* from the context is not null and not false, return the value of *expression* |
| (*name:expression*) | If value of key *name* from the context is null or is false, return the value of *expression* |
| (*name?exp1:exp2*) | If value of key *name* from the context is not null and not false, return the value of *exp1* otherwise return the value of *exp2* |
| (*name\*expression*) | If value of key *name* is Iterable or is an array expand the named template for each element in order |
| (*name\*template/expression*) | If value of key *name* is Iterable or is an array expand the named template for each element in order, separated by the value of *expression* |
| @*template* | Return the value of loading and expanding the named template using the current context |
| ‘*text*’ | Return the literal *text* |
| this | A pseudo context entry containing the current element when iterating |

**Portfolio** Part 2 – Web Site Generator

Create a stand-alone application which uses your Template Processing Library from part 1. This application should:

* Allow a user to specify (via command-line parameters, user input, or other appropriate means) *source*, *destination* and *templates* folders.
* Load all the templates from the *templates* folder into your template processor.
* Recursively traverse all the files and folders from the *source* folder performing the following actions:
  + For each folder, create an identical folder in the *destination* structure.
  + For each file
    - Load it as a collection of named values, for example by using a “properties file”.
    - Use the value of the name $template to find the appropriate template.
    - Call your template processing library to “expand” the template with these values.
    - Write the expanded string to the *destination*, with the same name and folder as where it came from in the *source*.