Specifications

The template engine is a java class that transforms template strings into "instanced" strings. For example, in the template string "Hello ${name} ${surname}" there are two templates:

1. ${name}
2. ${surname}

In order to do that, a template/value map must be provided. For example, if the template/value map contains two entries (name="Adam" and surname="Dykes") and the template string is "Hello ${name} ${surname}" the result would be: "Hello Adam Dykes".

Below there is the same example but in actual java code (all arguments will be specified):

```
1. EntryMap map = new EntryMap();
2. TemplateEngine engine = new TemplateEngine();
3. map.store("name", "Adam");
4. map.store("surname", "Dykes");
5. String result = engine.evaluate("Hello ${name} ${surname}", map,
   TemplateEngine.DELETE_UNMATCHED|TemplateEngine.CASE_SENSITIVE|TemplateEngine.ACCURA
   TE_SEARCH);
6. // The variable result contains the instanced string "Hello Adam Dykes".
```

The EntryMap class acts as a template/value map and the TemplateEngine class is the template engine which creates the "instanced" string.

1. EntryMap Class Specification

EntryMap is a java class responsible for tracking the values of the templates to be replaced in template strings.

This class exposes three methods for inserting, deleting and updating a template/value pair.

In this specification file, a template/value pair is called an "entry".

1.1 void EntryMap.store(String, String)

EntryMap.store Arguments:

arg1 (Type: String) - The template to be searched in the template string.

arg2 (Type: String) - The value to replace the template.

EntryMap.store Return Type:

Type: Void - This method does not return anything.

EntryMap.store Specifications:

- spec1 - Template cannot be NULL or empty. A runtime exception is thrown otherwise.
- spec2 - Replace value string cannot be NULL. A runtime exception is thrown otherwise.
- spec3 - The entries are ordered and follow the order in which they appear in the program.

In the previous example the "name"/"Adam" entry is the first entry since it was stored first.

- spec4 - Entries that already exist cannot be stored again.

In the following commands only the first entry will be saved:

```
1.  map.store("name", "Adam");
2.  map.store("name", "Adam");
```

1.2 void EntryMap.delete(String)

EntryMap.delete Arguments:

arg1 (Type: String) - The template from which value pair needs to be deleted.

EntryMap.delete Return Type:

Type: Void - This method does not return anything.

EntryMap.delete Specifications:

- spec1 - Template cannot be NULL or empty. A runtime exception is thrown otherwise.
- spec2 - After deleting a value pair, other remaining entries are still ordered as before.
- spec3 - Only existing value pair can be deleted. Otherwise nothing would happen.

1.3 void EntryMap.update(String, String)

EntryMap.update Arguments:

arg1 (Type: String) - The template from which value pair needs to be updated.

arg2 (Type: String) - The new replace value.

EntryMap.update Return Type:

Type: Void - This method does not return anything.

EntryMap.update Specifications:

- spec1 - Template cannot be NULL or empty. A runtime exception is thrown otherwise.
- spec2 - New replace value string parameters cannot be NULL. A runtime exception is thrown otherwise.
- spec3 - This operation does not change existing order.
- spec4 - Only existing value pair can be updated.

In the following commands only the first update operation works while the second one does not do anything:

```
1.  map.store("name", "Adam");
2.  map.update("name", "David");
3.  map.update("age", "23");
```

2. TemplateEngine Class Specification

TemplateEngine is a Java class which, given a template string and an EntryMap object, is able to produce the "instanced" string.

This class exposes only ONE non-static method (evaluate) which transforms the template string into an "instanced" string.

2.1 String TemplateEngine.evaluate(String, EntryMap, Integer)

TemplateEngine.evaluate Arguments:

arg1 (Type: String) - The template string to be instantiated.

arg2 (Type: EntryMap) - An EntryMap object containing the ordered entries.

arg3 (Type: Integer) - Defined as constant values and can be used jointly. Available matching mode includes:

TemplateEngine.DELETE_UNMATCHED

TemplateEngine.KEEP_UNMATCHED (Default)

TemplateEngine.CASE_SENSITIVE

TemplateEngine.CASE_INSENSITIVE (Default)

TemplateEngine.BLUR_SEARCH

TemplateEngine.ACCURATE_SEARCH (Default)

TemplateEngine.evaluate Return Type:

Type: String - This method returns the "instanced" string.

TemplateEngine.evaluate Specifications:

- spec1 - The template string can be NULL or empty. If template string NULL or empty, then the unchanged template string is returned.
- spec2 - The EntryMap object can be NULL. If EntryMap object NULL, then the unchanged template string is returned.
- spec3 - Three pairs of matching mode can be provided together using the operator |. Matching mode that is not specified will use the default value. Although the operator | is for logic OR computation, it is used here for to combine difference matching modes and you can consider it as “and” here. For example,

If the matching mode is 0 or NULL or DEFAULT or unsupported values, then KEEP_UNMATCHED, CASE_INSENSITIVE and ACCURATE_SEARCH will be used.

If the matching mode is DELETE_UNMATCHED, then DELETE_UNMATCHED, CASE_INSENSITIVE and ACCURATE_SEARCH will be used because the last two are default.

If the matching mode is CASE_SENSITIVE **|** BLUR_SEARCH, then KEEP_UNMATCHED, CASE_SENSITIVE and BLUR_SEARCH will be used.

When contradictory matching mode is set, the non-default one will be used. For example, if the matching mode is DELETE_UNMATCHED | KEEP_UNMATCHED | BLUR_SEARCH, then DELETE_UNMATCHED, CASE_INSENSITIVE and BLUR _SEARCH will be used. The reason is that DELETE_UNMATCHED and KEEP_UNMATCHED are contradictory, the program uses the non-default one DELETE_UNMATCHED, BLUR_SEARCH is picked by the user and CASE_INSENSITIVE is the default value of case sensibility.

If the matching mode is DELTE_UNMATCHED|KEEP_UNMATCHED|CASE_SENSITIVE, then ACCURATE_SEARCH, DELTE_UNMATCHED and CASE_SENSITIVE will be used.

- spec4 - Templates in a template string occur between "${" and "}". In a template, everything between its boundaries ("${" and "}") is treated as normal text when matched against an entry.

In the template string "Hello ${name}, could you please give me your ${item} ?" the two templates are:

1. ${name}
2. ${item}

The text of each template that will be matched against the EntryMap stored entries are:

1. "name"
2. "item"

(i.e. the template boundaries are omitted)

- spec5 - When a template is matched against an entry key and BLUR_SEARCH is enabled, any non-visible character does not affect the result.

The entry "middle name"/"Peter" will match all of the following templates:

1. ${middle name}
2. ${middlename}
3. ${middle name}

However, if ACCURATE_SEARCH is enabled, only ${middle name} can match the entry "middle name"/"Peter".

- spec6 - When CASE_INSENSITIVE is enabled, letter case is not taken in consideration when matching against entries.

The entry "name/Peter" can match (not limited to) the following templates:

1. ${Name}
2. ${naME}

However, if CASE_SENSITIVE is enabled, only ${name} can match the entry "name/Peter".

- spec7 - In a template string every "${" and "}" occurrence acts as a boundary of at MOST one template. - see evaluateBoundary() test

Processing from left-to-right, each "}" occurrence that is not already a boundary to a template is matched to its closest preceding "${" occurrence which also is not already a boundary to a template.

In the template string "I heard that }: ${name} said: ${we should try or best for winning the ${competition} cup.}" the templates are:

1. ${name}
2. ${competition}
3. ${we should try or best for winning the ${competition} cup.}

- spec8 - In a template string, different templates are ordered according to their length. The shorter templates precede.

In the case of same-length templates, the one that occurs first when traversing the template string from left-to-right precedes.

In the template string "abc}${de}${fgijk${lm}nopqr}${s}uvw${xyz" the sorted templates are:

1. ${s}

2. ${de}
3. ${lm}
4. ${fgijk${lm}nopqr}

- spec9 - The engine processes one template at a time and attempts to match it against the keys of the EntryMap entries until there is a match or the entry list is exhausted.

The engine processes both templates and entries according to their order.

If there is a match:

1. The template (including its boundaries) in the template string is replaced by the value of the matched entry.
2. The same replace happens to all other templates which include the replaced template.
3. The template engine moves on to the next template and repeats.

If the entry list is exhausted and no match found for the current template:

1. The template engine just moves on to the next template if KEEP_UNMATCHED is enabled.
2. The engine deletes the unmatched template from the template string and all other templates which include it.

2.2 Template Evaluation Example

Consider the following java example:

```
1.  EntryMap map = new EntryMap();
2.  TemplateEngine engine = new TemplateEngine();
3.  map.store("name", "Adam");
4.  map.store("surname", "Dykes");
5.  map.store("age", "29");
6.  String result = engine.evaluate("Hello ${name}, is your age ${age ${symbol}}", map,
     TemplateEngine.DELETE_UNMATCHED|TemplateEngine.BLUR_SEARCH);
```

In this example the ordered entries are:

1 - "name"/"Adam"

2 - "surname"/"Dykes"

3 - "age"/"29"

The ordered templates are:

1 - ${name}

2 - ${symbol}

3 - ${age ${symbol}}

The matching process is described step by step bellow:

step1 - template1 (${name}) is matched against entry1 ("name"/"Adam") ===> match!

---> Instanced string is now: "Hello Adam, is your age ${age ${symbol}}"

step2 - template2 (${symbol}) is matched against entry1 ("name"/"Adam") ===> no match!

---> Moving on to the next entry.

step3 - template2 (${symbol}) is matched against entry2 ("surname"/"Dykes") ===> no match!

---> Moving on to the next entry.

step4 - template2 (${symbol}) is matched against entry3 ("age"/"29") ===> no match!

---> Entry list exhausted and no match was found for template2.

step5 - matching mode "delete unmatched" is enabled so template2 is deleted from the instanced string and all other templates containing it.

---> Instanced string is now: "Hello Adam, is your age ${age }"

---> template3 is now: ${age }

step6 - template3 (${age }) is matched against entry1 ("name"/"Adam") ===> no match!

---> Moving on to the next entry.

step7 - template3 (${age }) is matched against entry2 ("surname"/"Dykes") ===> no match!

---> Moving on to the next entry.

step8 - Provided that matching mode "blur search" is enabled so template3 (${age }) is matched against entry3 ("age"/"29") ===> match!

---> Instanced string is now: "Hello Adam, is your age 29"

step9 - no more templates to process, algorithm terminates.

---> Returned instanced string: "Hello Adam, is your age 29"

3. SimpleTemplateEngine Class Specification

SimpleTemplateEngine is a Java class that is a simplified version of TemplateEngine. Users do not need to provide a separate EntryMap when they only want to replace only one same pattern in the template string. For example,

```
1.  engine = new SimpleTemplateEngine();
2.  String template = "Hi, my name is David. David is my forename.";
3.  String result = engine.evaluate(template, "David", "Peter", SimpleTemplateEngine.DE
    FAULT_MATCH);
```

Then the content of the string result is

Hi, my name is Peter. Peter is my forename.

You can specify which pattern in the template string is to be replaced. For example,

```
1.  String result = engine.evaluate(template, "David#2", "Peter", SimpleTemplateEngine.
    DEFAULT_MATCH);
```

Then the content of the string result is

Hi, my name is David. Peter is my forename.

This class exposes only ONE non-static method (evaluate) which transforms the template string into an "instanced" string.

3.1 String SimpleTemplateEngine.evaluate(String, String, String, Integer)

SimpleTemplateEngine.evaluate Arguments:

arg1 (Type: String) - The template string to be instantiated.

arg2 (Type: String) - Formatted pattern: The word to be replaced

arg3 (Type: String) - The word to replace the pattern

arg4 (Type: Integer) - Matching mode: Defined as constant values and can be used jointly. Available matching mode includes:

SimpleTemplateEngine.WHOLE_WORLD_SEARCH

SimpleTemplateEngine.CASE_SENSITIVE

SimpleTemplateEngine.DEFAULT_MATCH (default)

SimpleTemplateEngine.evaluate Return Type:

Type: String - This method returns the "instanced" string.

SimpleTemplateEngine.evaluate Specifications:

- spec1 - The template string can be NULL or empty. If the template string is NULL or empty, then the unchanged template string is returned.
- spec2 - The Formatted pattern can be NULL or empty. If the formatted pattern is NULL or empty, then the unchanged template string is returned.
- spec3 - The value string can be NULL or empty. If the value string is NULL or empty, then the unchanged template string is returned.
- spec4 - In the pattern string, '#' is considered as a special value to specify the pattern to be replaced. For example,

1. "David" - All the words "David" will be replaced.

2. "David#3" - Only the 3rd word "David" will be replaced. If there is not or less than 3 "David", nothing will happen.

3. "David##" - All the words "David#" will be replaced. "##" in the pattern is used as a single "#". "####" is considered as "##" etc.

4. "David###2" - Only the 2nd word "David#" will be replaced.

5. "Davi#3d" – Only the 3rd word "Davi" will be replace because after detecting the number 3, the remaining part of the pattern string is dropped.

- spec5 - The default matching mode (SimpleTemplateEngine.DEFAULT_MATCH) is not case sensitive and the pattern can be either a word or part of the word.

SimpleTemplateEngine.CASE_SENSITIVE enables case sensitive match

SimpleTemplateEngine.WHOLE_WORLD_SEARCH restrict that the pattern to be replaced should be a separate word. If it is NOT enabled, the following replacement can happen (pattern: "local", value string: "global"):

"localVARIABLE int localId = local" -> "globalVARIABLE int globalId = global"

If it is enabled, the example would be

"localVARIABLE int localId = local" -> "localVARIABLE int localId = global"

All special characters other than digits and letters are considered as word separators.

SimpleTemplateEngine.CASE_SENSITIVE and SimpleTemplateEngine.WHOLE_WORLD_SEARCH can be used together using SimpleTemplateEngine.CASE_SENSITIVE | SimpleTemplateEngine.WHOLE_WORLD_SEARCH. Recall that in our implementation, | is considered as “and”.

- spec6 - After a pattern string is replaced, the program continues to match pattern from the end of the replaced value. No recursive replacement should occur. For example, if we have "abc" as pattern and "abcabc" as value string, the template string "defabc" is changed to "defabcabc" rather than and endlessly and recursively replacing "abc"s with "abcabc".