ClubUML

CSYE7945 Spring 2013

Documentation for Similarity Checker (controller.similaritycheck)

Dong Guo

Chris Serrano

Long Chen

03/30/2013

Revision: 2.0

Table of Contents

[Revision History 3](#_Toc352401004)

[Introduction 4](#_Toc352401005)

[Four comparing functions 5](#_Toc352401006)

[Lowercase and Uppercase Check 5](#_Toc352401007)

[Spelling Error Check 6](#_Toc352401008)

[Singularity and Plurality Check 7](#_Toc352401009)

[Synonym Check 8](#_Toc352401010)

[Package API 10](#_Toc352401011)

[Design Diagrams 11](#_Toc352401012)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Author(s)** | **Notes** |
| 1.0 | 03/23/2013 | Dong Guo | The First Doc Draft |
| 1.1 | 03/24/2013 | Chris Serrano | Reviewed and edited |
| 2.0 | 03/30/2013 | Chris Serrano  Long Chen | Incorporated Long Chen’s documentation into this file |

# Introduction

The old comparing functions we got from last semester would only find exact matches between sets of Strings. Similarly named classes would now be acknowledged by the comparing functionality. For example, class “Bicycle” would not match “Bike”.

The goal of this doc is to help developers to have an understanding of the name-comparing functions that we created in package: controller.similaritycheck.

The functions inside provide a much more accurate feature that can find similar names, then provide better suggestions to users and help users to arrive at a final design for a diagram, after uploading different versions of the diagram to our software.

There are four basic comparing functions now. Considering about future maintenance and update we put them into four different Java files:

1. LowerUpperCheck.java
2. SpellCheck.java
3. PluralCheck.java
4. SynonymCheck.java

# Four comparing functions

In this section we will talk about the functions with more details.

There are only some small pieces of codes to give you a basic idea. Check SVN to see the entirety of the Java code.

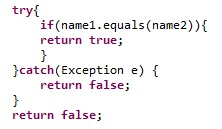
All the functions can be invoked separately, or use Package API to invoke them sequentially (see next section).

## Lowercase and Uppercase Check

1. **Description**: Compare two strings to see there is only a lowercase/uppercase problem.
2. **Public API**: isSimilarWord(String name1, String name2)
3. **Return**: boolean
4. **Related files:** LowerUpperCheck.java
5. **Major logic**:
6. No matter what input we have, use toLowerCase() to change them into lowercase.

C:\Users\Dong Guo\AppData\Roaming\Tencent\Users\148309662\QQ\WinTemp\RichOle\%OSFQARD`%M7CG(04TV]GDJ.jpg

1. Then compare. If they are the same, return true, otherwise return false.

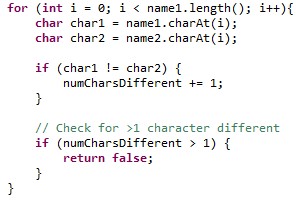


1. **Example**:“bike” and “BIKE” are similar.

## Spelling Error Check

1. **Description**: Compare two strings to see there is only a spell error. One [character](app:ds:character) or a transposed problem.
2. **Public API**: isSimilarSpelling(String name1, String name2)
3. **Return**: boolean
4. **Related** **files**: SpellCheck.java
5. **Major logic**:

a) Part 1: Iterate through characters of input Strings to see if there is only one character that is not the same between the two Strings.



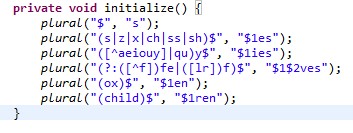
b) Part 2: Set two marks, “diffFlag” and “transposed”. Store the first two different characters, use the two marks to check if they have opposite positions and they are the only different characters through the entire string.

1. **Example**:
   1. For Part 1, “Northeastern” and “Mortheastern” are similar.
   2. For Part 2, “project” and “proejct” are similar.

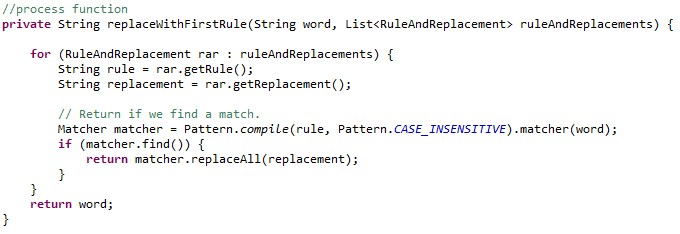
## Singularity and Plurality Check

1. **Description**: Compare two strings to see if they are the same except for one being singular and the other being plural. Since this is not the core of our project and it’s really hard to make it work for all words, now it just works for most words. Other special cases of less common plural forms of words would be useful as a future update but is not considered a high priority for this semester.
2. **Public API**: isSimilarNoun(String name1, String name2)
3. **Return**: boolean
4. **Related** **files**:
   1. PluralCheck.java
   2. PluralInflector.java
5. **Major logic**: Use Plural Inflector to change the shorter string to its plurality. Then compare the two strings to see if they are the same.

a) Use regular expression to create different patterns. The advantages of doing this is, in the future, if we want to make it more comprehensive, we can just add new patterns.



b) If we can find a matched pattern, we replace the part and then return the new word



c) In isSimilarNoun(), we simply do: PluralInflector.getInstance().pluralize(String name)

1. **Example**:
   1. “project” and “projects” are similar.
   2. “child” and “children” are similar.
   3. “wife” and “wives” are similar

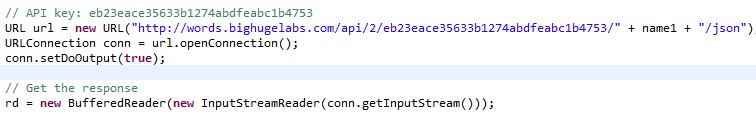
## Synonym Check

1. **Description**: Compare two strings to see if they are synonym to each other.
2. **Public API**: isSimilarThesaurus(String name1, String name2)
3. **Return**: boolean
4. **Related** **files**: SynonymCheck.java
5. **Major logic**:

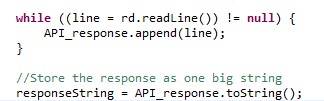
a) Use a public API from bighugelabs.com to get a list of the synonyms of one string. Here we need an API key which is: eb23eace35633b1274abdfeabc1b4753

The account username: [guo.do@husky.neu.edu](mailto:guo.do@husky.neu.edu)

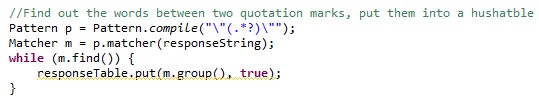
The account password: umlclub2013



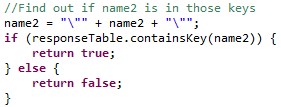
b) Read and store it line by line in a StringBuilder. Then make it into a big string.



c) Use regular expression to get out all the words we want. And put them into a hashtable, set the value to true.



d) Use another name string to check the hashtable.



1. **Example**:
   1. “project” and “program” are similar.
   2. “baby” and “honey” are similar.

# Package API

1. **Description**: Compare two strings to see if they are similar.
2. **Public API**: doSimilarityCheck()
3. **Return**: boolean
4. **Related** **files**:

SimilarityCheck.java

LowerUpperCheck.java

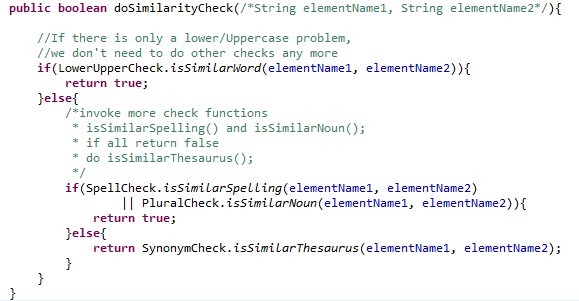
SpellCheck.java

PluralCheck.java

PluralInflector.java

SynonymCheck.java

1. **Major logic**:
   1. Do isSimilarWord() first, if the return is true, break out
   2. Otherwise do isSimilarSpelling() and isSimilarNoun(), if the return is true, break out
   3. In the end, do isSimilarThesaurus()



1. **Testing.java:**

The testing.java is created to test all the functions inside the package.

# Design Diagrams

Figure 1 shows the class diagram for the entire similarity check package. Figure 2 shows the sequence diagram of a user calling the public method SimilarityCheck.doSimilarityCheck() and the sequence of messages as it conducts the various tests.

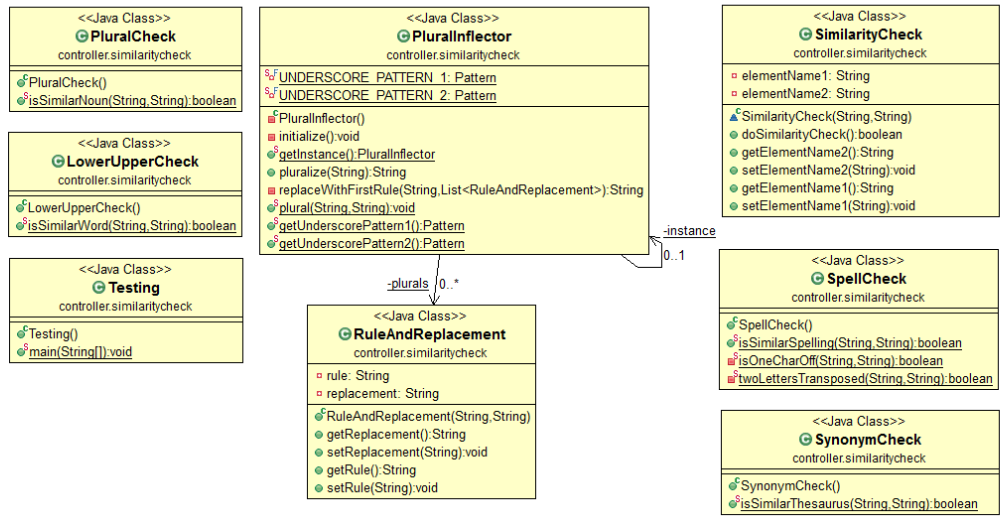


Figure : Similarity Package Class Diagram

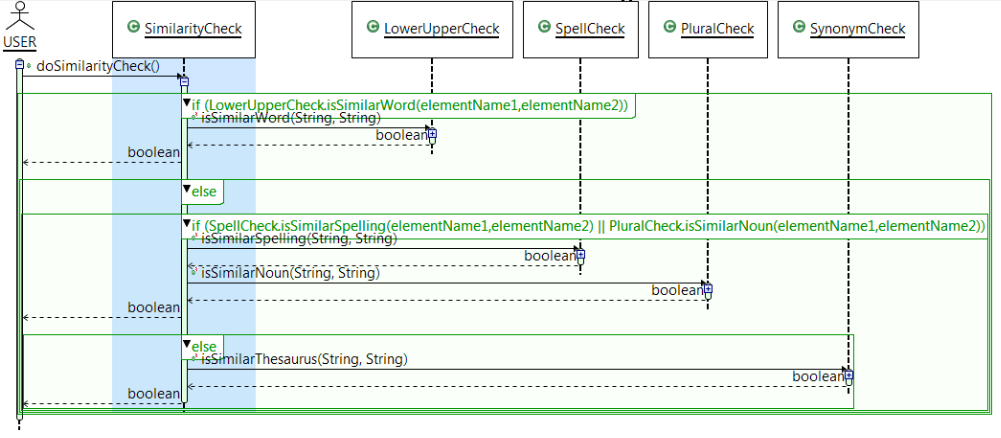


Figure : Similarity Package Sequence Diagram