Flow SOLID

Nonfunctional requirements – Technically robust systems

You must work in pairs. The assignment consists of two parts. Each hand in must consist of one zip-file containing:

- An zip'ed NetBeans project
- A word- or pdf-file with other artifacts as stated in this text

Idea:

You must develop a program for which the requirements can be changed. One key objective of this assignment is to demonstrate the benefits of following recognized guidelines for object oriented design. Hence, the focus will be on achieving the following program qualities

- Maintainability
- Extensibility
- Reusability
- Testability

Part 1: Language Trainer

Write a program that supports the task of learning words in a foreign language.
Design and implement the program in steps – as follows.

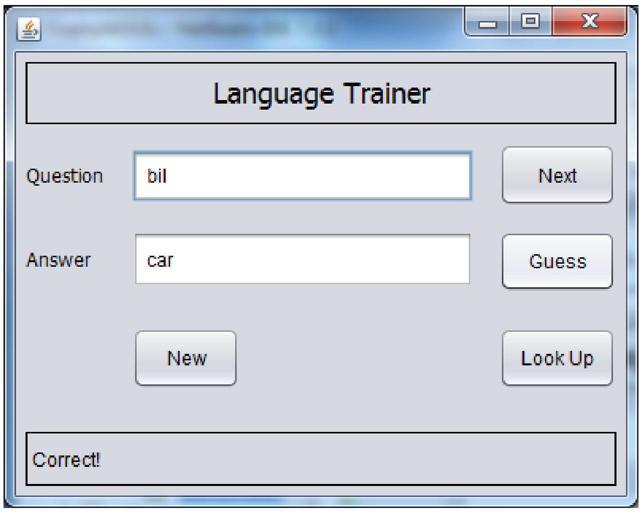
Green, Yellow & Red: Basic funcionality

The user must be able to enter new ”challenging” pairs of words (i.e. danish/english). The program must store these word pairs.

Then the program can present a ”task” by randomly selecting and displaying a word in one language and let the user enter the corresponding word in the other language.

The program gives feedback (right/wrong).

Example GUI



Question: bil
Answer: car
Correct!

Green, Yellow & Red: **File I/O**

The program must be able to read a set of word pairs (a ”language package”) from a text file and correspondingly save all word pairs in the same file. The format of the file might be:

```
horse, hest
building, bygning
```

Green, Yellow & Red: **Dictionary**

The user must be able enter a word in one of the languages and let the program look up the corresponding word in the other language.

Constraints

- The program *must* contain a Control class that *implements* a predefined Java Interface: **`WordPairControlInterface.java`** (see separate file). This requirement is *mandatory and crucial*.
- You *must test* the methods of the Control class using a predefined *JUnit-test* (handed out).
- Near the end of part 1, you must *swap GUI with another group* and thereby demonstrate compliance with the interface contract.
- You must maintain a rough *design class diagram* (DCD) showing the *overall structure* of the program. This will serve as a visual common reference for co-operation and guidance.

Yellow and red User adaptation

Assume that the user improves his/her skills. Now, we want the program to automatically adapt to the current level of the user.

Create a mechanism that continuously changes the probability of the selection of a given word pair for the next task. The probability should decrease when the user gives a correct answer and increase when the user gives a wrong answer.

Hint: Changing probabilities may be implemented in many ways. One possible strategy may be to assign a priority to each word pair (ex: a number between 1 and 5) and change this number according to the correctness of a guess. On selecting the next word for a ”task” a random number selected from another wider range (ex: 1 to 15) could be used as follows:

Ex:

Random number ➔ select word with priority

1-5	➔	1
6-9	➔	2
10-12	➔	3
13-14	➔	4
15	➔	5

Hand in (part 1):

Format: One (1!) zipped file

Content:

- The program - as a zipped NetBeans project
- Use Case diagram
- Design Class Diagram
- Sequence Diagram
- Short note on
  - the responsibility of the most important class(es)
  - compliance with general OO design guidelines (See Larman hand-out)
  - your observations during the test of the GUI from another group
  - the result of the JUnit test
- Your proposals – if any – for a more appropriate interface (`WordPairControlIF.java`)

Learning Objectives - flow Solid:

Purpose: Be able to develop a technically robust program that comply with general guidelines for object oriented design

Skills: Be able to

- use an interface as a contract between development groups
- write a class that uses or implements an interface
- write simple search algorithms (linear search)

- use the test tool JUnit
- define relevant test cases for a method

- place responsibility on classes appropriately
- evaluate design alternatives according to OO design criteria
  (Low Coupling, High Cohesion, Information Expert)

- differentiate between and use structural and behavioral (dynamic) diagrams
- use UML diagrams (DCD and SD)
- design static structures (classes) that correlate to interactions diagrams (SD)

- use relevant and correct technical language in oral communication

Ressources:

Interfaces, polymorphism, searching:
L&L: chap 7.5, 10.1, 10.3, 10.5
http://docs.oracle.com/javase/tutorial/java/concepts/interface.html
http://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html

Test:
Tremblay & Cheston: Data Structures and Software Development,Prentice Hall 2003, ISBN 137879539 16-16.5 (copy on Fronter)

JUnit:
http://junit.org/+
J.B.Rainsberger: JUnit Recepies: chap 1
http://tech.sxinfo.net/UserFiles/mxk/File/eBook/java/junit/Manning%20-%20JUnit%20Recipes%20-%202005.pdf

UML Design Class Diagram , UML SequenceDiagram:
UML 2.0 In A Nutshell: kap 2 (11-15, 19-20, 24-32) kap 10 (128-136, 138-141, 147-148)

Object Oriented Design:
L&L chap 7-7.4
MSDN Magazine: Cohesion And Coupling
http://msdn.microsoft.com/en-us/magazine/cc947917.aspx

```
package interfaces;

/**

 *

 * @author CHU

 */


public interface WordPairControlInterface

{


/**

    * Pre: Post: A new word pair is added to the existing collection of word

    * pairs

    */

   void add(String question, String answer);


   /**

    * Pre: Post: Returns the number of wordpairs

    */

   int size();


   /**

    * Pre: At least one word pair must be present Post: Returns a question

    * randomly selected from the collection of word pairs

    */

   String getRandomQuestion();


   /**
```

```
     * Pre: Post: Returns true if (question, quess) exists as a word pair in the
     * collection, otherwise false
     */
    boolean checkGuess(String question, String quess);

    /**
     * Pre: Post: Returns the answer corresponding to the question if this
     * exists in the collection. Otherwise it returns null.
     */
    String lookup(String question);

    /**
     * Pre: Post: Word pairs are read from the file â€œfilenameâ€� and added to the
     * collection of word pairs. Returns true if successfully done. Otherwise it
     * returns false.
     */
    boolean load(String filename);

    /**
     * Pre: Post: All word pairs from the collection has been written to the
     * file â€œfilenameâ€� Returns true if successfully done. Otherwise false.
     */
    boolean save(String filename);

    /**
     * Pre: Post: The existing collection of word pairs is cleared
     */
    void clear();
}
```