OOB.03 — Vocabulary Trainer

Table of Contents

This time you are going to implement a vocabulary trainer. It will only be terminal based, but we'll use some fancy colors and headings at least.

```
@startuml

class Trainer {
    -int WORD_IDX [const]
    -int TRANSLATION_IDX [const]
    -int CYCLE_COUNT [const]
    -VocabularyItem[] _vocabularyItems [readonly]

    +Trainer(string[][])
    +void PerformTrainingCycle()
    +void PrintStatistics()
    -int PickNextWord(bool[])
    -void Sort()
    {static} -VocabularyItem[] CreateVocabularyItems(string[][])
}

class VocabularyItem {
    -int _countCorrect
    -int _countAsked
    +string NativeWord [readonly]
    +string Translation [readonly]

    +VocabularyItem(string, string)
    +bool TestTranslation(string)
    +int CompareTo(VocabularyItem)
    +string ToString() [override]
    {static} -int CompareStrings(string, string)
}

Trainer "1" -r- "n" VocabularyItem: has

@enduml
```

Some parts of the application have already been provided to you, especially the `Program.cs` file.

You might need to comment some code to allow the program to compile initially

1. `VocabularyItem` class

This class is meant to hold one 'word':

1. The 'native' word
2. The translation

For example: 'horse' & 'Pferd'.

It also holds information about the 'translation success' of the user who is currently training:

- How often was user 'asked' to translate this specific word
- How often did the user get it right (correct translation)

The following operations are supported:

- Test a user provided translation
  - Returns `true` or `false` accordingly
  - Updates the internal statistics
- Compare to another instance of `VocabularyItem`
  - Used for sorting before displaying the statistics
  - First we compare the number of correct answers.
    If those are equal we then compare the native words lexically.
- Create a `string` representation of the `VocabularyItem` instance
  - This method requires the `override` keyword, we'll talk about that later. For now just put it there and be happy that everything works.
- Compare two strings lexically while ignoring case
  - This is an internal helper method
  - Look up `string.Compare` & `StringComparison.OrdinalIgnoreCase` in the documentation for the implementation of this method

▼ *Cheating: Look at the XMLDoc of the methods if you have a hard time understanding what each is supposed to do*

```
/// <summary>
///     A translation attempt is checked for correctness.
/// </summary>
/// <param name="translationAttempt">The user provided translation</param>
/// <returns>True if the translation was correct; false otherwise</returns>

/// <summary>
///     The vocabulary item is compared to another. First the number of correct
```

```
answers is compared.
///     If it is equal the native words are compared ordinal.
/// </summary>
/// <param name="other">The <see cref="VocabularyItem"/> to compare with</param>
/// <returns>0 if equal; less than 0 if this item is smaller; greater than 0
otherwise</returns>

/// <summary>
///     Overrides the default string representation to display the word and
translation statistics.
/// </summary>
/// <returns>A string containing the word, its translation and the training
statistics</returns>

/// <summary>
///     Compares two strings by ordinal value, ignoring case.
/// </summary>
/// <param name="a">First string</param>
/// <param name="b">Second string</param>
/// <returns>Less than 0 if a precedes b in the sorting order; greater than 0 if b
precedes a; 0 otherwise</returns>
```

2. `Trainer` class

This class contains the vocabulary and is responsible for guiding the user through a training cycle.

A considerable part of the 'user interface' is handled by `Program`, be aware which class performs which action(s) based on its purpose

It provides the following operations:

- Turn 'raw' words read from the `vocabulary.csv` file (provided as `string[][]`) into an array of `VocabularyItem` instances
  - Invalid entries (native word, translation or both are `null` *or empty*) are skipped ⇒ the final array is trimmed as usual (or rather, a temporary array can be used)
    - That's easily possible with two loops in total (within this method)
  - Make use of the `WORD_IDX` & `TRANSLATION_IDX` constants
- Sort the vocabulary
  - The sorting happens in-place
    - Use any algorithm you want
  - Called before printing the statistics
  - Make use of the `CompareTo` method of the `VocabularyItem` class
- Randomly pick the next word (from the vocabulary) to show to the user
  - In general, words are picked by random. However, a word should not reappear while

another hasn't been used yet. To implement this logic use the `bool[]` in which you can track which vocabulary items have already been used (via index). Draw a new random number until you get an 'unused' word. **Once all words have been used at least once any one is chosen.**

        - Keep in mind: an array parameter is passed by *reference* ⇒ useful here
    - Make sure to use the `RandomProvider` otherwise you'll have a *very hard* time writing your unit tests
- Print training statistics
    - Already implemented for you
- Perform a training cycle
    - *Partially* implemented already
    - Look at the sample run to learn about the expected interaction and output format
    - Make sure to call the `TestTranslation` method of the *proper* `VocabularyItem` *instance* when checking translation input

3. Sample Run

[] | *Sample Run*