LABORATION 4

Utveckla en applikation för att träna glosor

Inledning

Du får i uppdrag att utveckla ett program där man ska kunna skapa och redigera gloslistor på olika språk, samt kunna öva på glosor från listorna man lagt in.

Programmet ska finnas i två versioner: en version som körs i terminal (consoleapp), samt en version med GUI (winforms). De båda applikationerna ska dela kod i ett gemensamt bibliotek (class library).
OBS! Använd .NET 5 (Core) för samtliga projekt

Uppdragsgivaren vill även kunna använda biblioteket för ett webprojekt som de utvecklar parallellt, och har därför strikta riktlinjer över vilka funktioner som ska implementeras.

Filformat

Varje ordlista ska sparas som en egen fil med filändelse **.dat**, och namnet på filen är namnet på listan. T.ex. 'mywords.dat'. Första raden i filen innehåller namn på språken, separerade med semikolon. Därefter följer en rad för varje glosa, med översättningarna separerade med semikolon. Se bifogad exempel-fil.

Alla .dat filer ska lagras i en mapp med samma namn som applikationen under användarens appdata\local. Om mappen inte redan finns måste applikationen skapa den.

Klassbibliotek (Class library)

Biblioteket ska implementera klasserna **Word** samt **WordList** enligt nedan:

Word

Klassen Word används för att lagra ett enskilt ord och ska ha egenskaper (properties) enligt nedan

Properties

public string[] Translations { get; }
public int FromLanguage { get; }
public int ToLanguage { get; }

Translations lagrar översättningarna, en för varje språk. Med FromLanguage och ToLanguage kan man ange för övningar vilket språk som ska översättas till respektive från. Dessa används av metoden **WordList.GetWordToPractice()**

Konstruktorer

Klassen ska ha även ha konstruktor i två versioner

public Word(params string[] translations)
initialiserar 'Translations' med data som skickas in som 'translations'

public Word(int fromLanguage, int toLanguage, params string[] translations)
som ovan, fast sätter även FromLanguage och ToLanguage.

WordList

Klassen WordList består av en lista av typ Word (dvs `List<Word>`), samt tillhandahåller metoder för att

- ladda in och spara listan från .dat
- lägga till och ta bort ord
- slumpa ord för övningar
- andra metoder beskrivet enligt nedan:

Properties:

public string Name { get; }	Namnet på listan.
public string[] Languages { get; }	Namnen på språken.

Metoder:

public Wordlist(string name, params string[] languages)
Konstruktor. Sätter properites Name och Languages till parametrarnas värden.

public static string[] GetLists()
Returnerar array med namn på alla listor som finns lagrade (utan filändelsen).

public static Wordlist LoadList(string name)
Laddar in ordlistan (name anges utan filändelse) och returnerar som WordList.

public void Save()
Sparar listan till en fil med samma namn som listan och filändelse .dat

public void Add(params string[] translations)
Lägger till ord i listan. Kasta ArgumentException om det är fel antal translations.

public bool Remove(int translation, string word)
translation motsvarar index i Languages. Sök igenom språket och ta bort ordet.

public int Count()
Räknar och returnerar antal ord i listan.

public void List(int sortByTranslation, Action<string[]> showTranslations)
sortByTranslation = Vilket språk listan ska sorteras på.
showTranslations = Callback som anropas för varje ord i listan.

public Word GetWordToPractice()
Returnerar slumpmässigt Word från listan, med slumpmässigt valda FromLanguage och ToLanguage (dock inte samma).

Console Application

Man ska kunna skapa listor, lägga till och ta bort ord, öva, m.m genom att skicka in olika argument till programmet. Om man inte skickar några argument (eller felaktiga argument) ska följande skrivas ut:

```
Use any of the following parameters:
-lists
```

```
-new <list name> <language 1> <language 2> .. <language n>
-add <list name>
-remove <list name> <language> <word 1> <word 2> .. <word n>
-words <listname> <sortByLanguage>
-count <listname>
-practice <listname>
```

-lists
Listar namnen på alla ordlistor från mappen i appdata/local/”mapp med .dat filer”

-new <list name> <language 1> <language 2> .. <language n>
Skapar (och sparar) en ny lista med angivet namn och så många språk som angivits. Går direkt in i loopen för att addera nya ord (se -add).

-add <list name>
Frågar användaren efter ett nytt ord (på listans första språk), och frågar därefter i tur och ordning efter översättningar till alla språk i listan. Sedan fortsätter den att fråga efter nya ord tills användaren avbryter genom att mata in en tom rad.

-remove <list name> <language> <word 1> <word 2> .. <word n>
Raderar angivna ord från namngiven lista och språk.

-words <listname> <sortByLanguage>
Listar ord (alla språk) från angiven lista. Om man anger språk sorteras listan efter det, annars sortera efter första språket.

-count <listname>
Skriver ut hur många ord det finns i namngiven lista.

-practice <listname>
Ber användaren översätta ett slumpvis valt ord ur listan från ett slumpvis valt språk till ett annat. Skriver ut om det var rätt eller fel, och fortsätter fråga efter ord tills användaren lämnar en tom inmatning. Då skrivs antal övade ord ut, samt hur stor andel av orden man haft rätt på.

Winforms application

Uppdragsgivaren har inga specifika önskemål på hur denna utformas mer än att den ska ha samma funktionalitet som återfinns i console appen. D.v.s det man kan göra i console appen ska också gå att göra i GUI:t.

Tips & Hjälp

- `Environment.GetFolderPath( Environment.SpecialFolder.LocalApplicationData)` returnerar sökvägen till användarens '/appdata/local' mapp
    - I Utforskaren heter mappen %LOCALAPPDATA%
- För hantering av filer och mappar, kolla upp följande klasser:
    - `Path, Directory, StreamWriter, StreamReader`
- Tänk på att göra både user input och lagrade data till lower case innan du jämför dem om du vill att ditt program ska vara case insensitive.
- Du kan skicka parametrar till din konsolapp på två sätt:
    1. Starta Kommandotolken och ställ dig i den mappen där ditt körbara program ligger.
    2. Skicka in via Properties för ditt project. Titta under Debug → Application Arguments. Varje argument tas emot i argumentet `string[] args` i metoden Main. Vissa argument har ett minustecken som första tecken.
- Namn på datafiler sätts lämpligen efter de språk som filen innehåller.
    - Använd samma språkbeteckningar som används av .NET CultureInfo, så som **sv-SE** för svenska och **en-GB** för brittisk engelska.
- För VG krävs att det finns ett klassdiagram för klassbiblioteket. Det är en god idé att börja med att skapa diagrammet, även om man siktar på G.

Redovisning

Inlämning skall göras individuellt.

Lämna in uppgiften på ithsdistans med en kommentar med GitHub-länken.

Betygskriterier

För godkänt:

- Klassbiblioteket ska innehålla metoder och egenskaper enligt specifikation.
- Båda programmen ska fungera med all funktionalitet på plats.
- Applikationerna ska vara testade och rimligt buggfria. Be gärna någon kompis att testa ditt program och försöka hitta buggar.
- För godkänt är det acceptabelt om man begränsar WordList så varje lista endast har stöd för två språk. Detta underlättar datahantering, såväl som inmatning i konsollappen och utformandet av GUI. Om man väljer att göra detta så kan ”params”-parametern i WordList konstuktor, i Add(), samt i Word konstruktor bytas mot två separata string-parametrar. I övrigt ska specifikationen följas.
- WordList får inte innehålla en *publik* lista av Word. Tanken är att man ska använda metoderna List() och GetWordToPractice() i applikationerna.
- **Utöver att skriva själva koden, ska ni även** föreslå minst 1 ny funktion / förbättring av apparna, samt ge en rimlig tidsuppskattning för implementation av denna. Skriv som en kommentar på uppgiften på ithsdistans.
- **Alla 3 projekt ska använda .NET 5**, (inte .NET Framework eller Core 3.x)
- **Lösningen ska vara incheckad korrekt på GitHub** Rotmappen ska alltså innehålla VS solution-filen (.sln) samt en mapp för varje (3) projekt. Var och en av dessa mappar ska innehålla en VS projektfil (.csproj) och alla projektets .cs filer (och .resx filer för winforms).

För väl godkänt krävs även:

- Koden ska vara väl strukturerad och lätt att förstå.
- Det skall finnas ett korrekt klassdiagram för klassbiblioteket
- Allting ska fungera vid inlämning. Gör det inte det så kommer ni få tillbaks uppgiften med möjlighet att fixa det som krävs för godkänt. Men ni har alltså bara en chans på er att få VG. Så var noga med testning.
- Inlämning sker före deadline.
- WordList har stöd för godtyckligt antal språk, enligt spec.

- Föreslå istället minst **3** nya funktioner / förbättringar samt ge en rimlig tidsuppskattning för implementation av vardera. Skriv som en kommentar på uppgiften på ithsdistans.
- Att man tänkt igenom användargränsnittet (winforms) så att det känns användbart och enkelt att jobba med.

Exempeldata

```
sv-SE;en-US
alldeles;quite
att;to
av;of
axel;axel
behörig;competent
besked;information
busig;mischievous
det;it
duktig;good
efter;after
elevassistent;teacher's assistant
en;a
för;for
han;he
har;has
hålla;keep
hög;high
i;in
inte;not
jag;I
jaga;chase
klass;class
lägga;lay
lärare;teacher
man;man
med;with
mycket;very
måste;must
nu;now
och;and
ordförande;chairman
pengar;money
pojke;boy
på;on
rätt;right
sak;thing
satsa;invest
```

```
schack;chess
se;see
ska;shall
skola;school
skriv;write
som;as
säga;say
underskott;deficit
vara;be
```