|  |
| --- |
| Laborationsanvisning |

Generera underlag till användarkonton

Steg 3, laborationsuppgift 1

|  |
| --- |
| *Författare:* Mats Loock  *Kurs:* Inledande programmering med C#  *Kurskod:*1DV402 |

**Innehåll**

[Uppgift 3](#_Toc318713249)

[Problem 3](#_Toc318713250)

[Format på textfil med anställda 3](#_Toc318713251)

[Algoritm för att läsa in anställda 4](#_Toc318713252)

[Klassdiagram 4](#_Toc318713253)

[Strukturen Employee 5](#_Toc318713254)

[Klassen User 6](#_Toc318713255)

[Klassen Repository 7](#_Toc318713256)

[Klassen AdService 8](#_Toc318713257)

[Klassen FakeActiveDiretory 9](#_Toc318713258)

[Klassen Program 9](#_Toc318713259)

[Krav 9](#_Toc318713260)

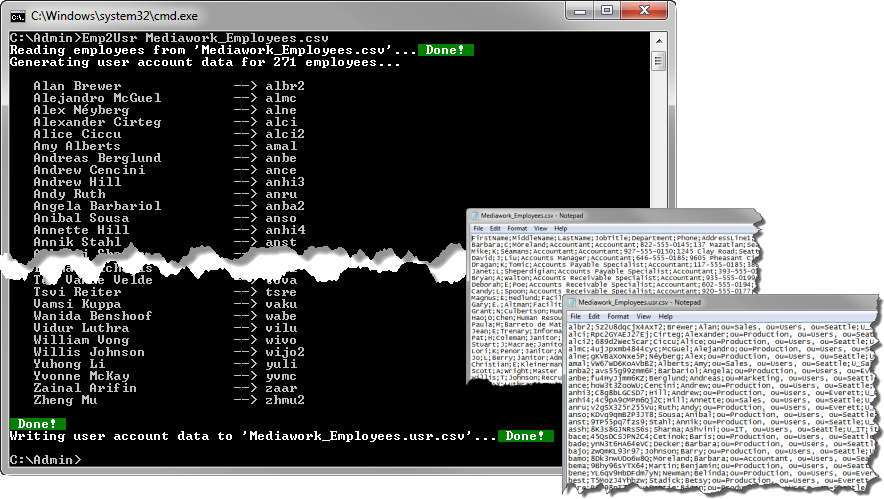
[Läsvärt 10](#_Toc318713261)

# Uppgift

## Problem

Med utgångspunkt från en textfil med flera hundra anställda på ett företag tillhörande olika avdelningar ska du färdigställa en påbörjad applikation som skapar en textfil innehållande det data som krävs för att AD-konton ska kunna skapas med hjälp av existerande batch-fil.

Applikationen ska vara utformad så att den kan köras från kommandoprompten utan att användaren behöver interagera med applikationen efter att applikationen startat.



Figur 1.

Applikationen ska delas upp i flera typer med klara ansvarsområden, t.ex. ska all hantering av filer placeras i en klass medan hanteringen av anställda och AD-användare placeras i andra klasser. Ett allmänt krav på applikationen är att den ska ge beskrivande felmeddelanden då fel eventuellt inträffar.

## Format på textfil med anställda

Information om de anställda finns att tillgå i form av en textfil. För att kunna skilja anställda åt är textfilen vara formaterad på ett bestämt sätt.

Varje anställd har en egen rad i textfilen där varje rad består av 12 delar separerade med semikolon (;). Första raden innehåller rubriker som definierar typen av data för de olika delarna.



Figur 2. Del av textfil med semikolonseparerade värden med information om anställda.

## Algoritm för att läsa in anställda

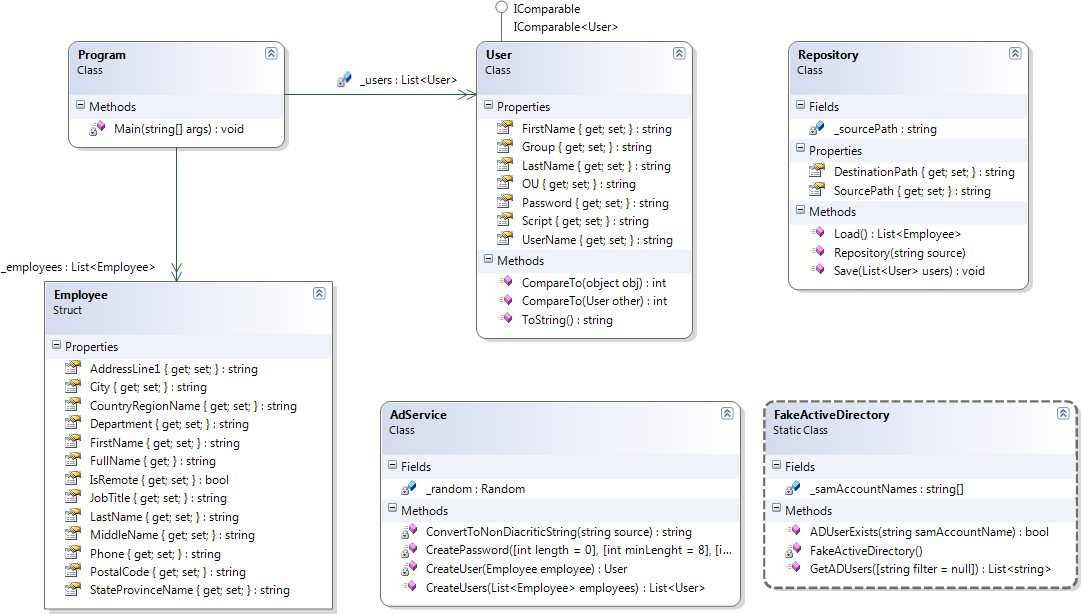
Textfilen med anställda ska läsas rad för rad. Varje rad ska tolkas för att bestämma vad raden beskriver. Förfarandet att läsa in och tolka textfilen till en samling med objekt kan upplevas vara en stor utmaning varför en beskrivning av en algoritm kan underlätta.

Algoritmen beskriver innehållet i metoden Repository.Load() som returnerar en lista med referenser till Employee-objekt.

1. Skapa lista som kan innehålla referenser till objekt representerande anställda.
2. Öppna textfilen för läsning.
3. Läs rad med rubrik från textfilen.
4. Läs rad från textfilen tills det är slut på filen.
   1. Om det är en tom rad…
      1. …fortsätt med att läsa in nästa rad.
   2. …annars är det en rad med information om en anställd
      1. Dela upp raden i delar genom att använda metoden Split() i klassen string. De olika delarna separeras åt med semikolon varför det alltid ska bli 12 delar.
      2. Om antalet delar inte är 12…
         1. …är något fel varför ett undantag ska kastas.
      3. Skapa ett objekt representerande den anställde och initiera det med de 12 olika delarna.
      4. Lägg till den anställde till listan med anställda.
5. Sortera listan med anställda med avseende på förnamn och efternamn.
6. Returnera en referens till listan.

## Klassdiagram

Applikationen ska delas upp i flera typer. Typerna Employee, User, Repository, AdService, FakeActiveDirectory och Program ansvara var och en för sin del av applikationen.



Figur 3. Övergripande klassdiagram där klassen FakeActiveDirectory samt metoderna ConvertToNonDiacriticString och CreatePssword i klassen AdService är givna.

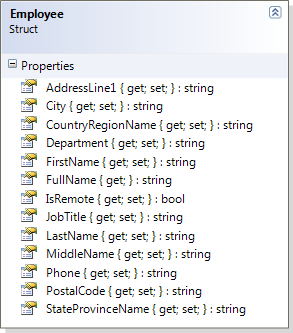
Strukturen Employee beskriver anställd med alla relevanta uppgifter. Klassen User definierar data intressant för att skapa en anställds användarkonto.

Repository ansvarar för allt som har med persistent lagring av anställda och underlag till användarkonton, d.v.s. klassen har metoder för att läsa anställda från en textfil och skriva underlag till användarkonton till en textfil.

Klassen Program har huvudansvaret för exekveringen av applikationen och ansvara för att tolka argument till applikationen, skapa lämpliga objekt och anropa metoder.

### Strukturen Employee

För att lagra information om en anställd ska en struktur användas. Strukturen ska vara enkelt utformad och bara ha autoimplementerade egenskaper, vilket innebär att ingen av egenskaperna ska vilideras. Den ska även innehålla en ”read-only”-egenskap för den anställdes fullständiga namn bestående av för- och efternamn.



Figur 4. Strukturen Employee.

#### Egenskapen AdressLine1

Publik autoimplementerad egenskap av typen string representerande adressen.

#### Egenskapen City

Publik autoimplementerad egenskap av typen string representerande orten.

#### Egenskapen CountryRegionName

Publik autoimplementerad egenskap av typen string representerande landet.

#### Egenskapen Department

Publik autoimplementerad egenskap av typen string representerande avdelningen.

#### Egenskapen FirstName

Publik autoimplementerad egenskap av typen string representerande förnamnet.

#### Egenskapen FullName

Publik ”read-only”-egenskap av typen string sammansatt av förnamnet och efternamnet.

#### Egenskapen IsRemote

Publik autoimplementerad egenskap av typen bool representerande om den anställde arbetar på distans.

#### Egenskapen JobTitle

Publik autoimplementerad egenskap av typen string representerande yrkestiteln.

#### Egenskapen LastName

Publik autoimplementerad egenskap av typen string representerande efternamnet.

#### Egenskapen MiddleName

Publik autoimplementerad egenskap av typen string representerande mellannamnet.

#### Egenskapen Phone

Publik autoimplementerad egenskap av typen string representerande ett telefonnummer.

#### Egenskapen PostalCode

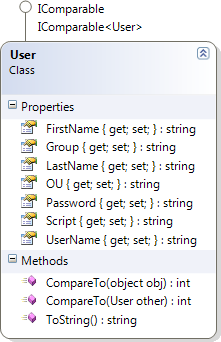
Publik autoimplementerad egenskap av typen string representerande postnumret.

#### Egenskapen StateProvinceName

Publik autoimplementerad egenskap av typen string representerande namn på staten.

### Klassen User

Instanser av klassen används för att representera en anställds underlag till användarkonto. Klassen implementerar IComparable och IComparable<T> för att det ska vara möjligt att sortera instanser med avseende på användarkontots namn.



Figur 5. Klassen User som implementerar två interface.

#### Egenskapen FirstName

Publik autoimplementerad egenskap av typen string representerande förnamnet.

#### Egenskapen Group

Publik autoimplementerad egenskap av typen string representerande den universella gruppens namn.

#### Egenskapen LastName

Publik autoimplementerad egenskap av typen string representerande efternamnet.

#### Egenskapen OU

Publik autoimplementerad egenskap av typen string representerande den organisatoriska enhet användaren ska tillhöra.

#### Egenskapen Password

Publik autoimplementerad egenskap av typen string representerande lösenordet.

#### Egenskapen Script

Publik autoimplementerad egenskap av typen string representerande namnet på logonskriptet.

#### Egenskapen UserName

Publik autoimplementerad egenskap av typen string representerande användarnamnet.

#### Metoderna CompareTo

CompareTo() ska överlagras, d.v.s. det ska finnas två metoder med samma namn men med olika parameterlistor.

Metoderna anropas i regel inte direkt av kod utvecklare skriver utan det sker automatiskt av ramverket. Metoden CompareTo(object obj) används t.ex. av metoden Array.Sort() då instanser av typen User ska sorteras. Metoden CompareTo(User other) används av metoden List.Sort() då instanser av typen User ska sorteras.

Metoderna ska jämföra två objekt med avseende på fältet för användarnamnet.

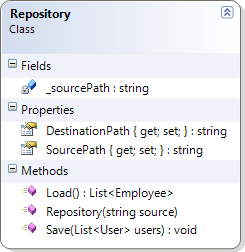
* Refererar parametern till null ska ett heltal större än 0 returneras.
* Refererar parametern till ett objekt som inte är av typen User ska ett undantag av typen ArgumentException kastas.
* Refererar parametern till ett objekt vars användarnamn ska sorteras efter det anropande objektets användarnamn ska ett heltal mindre än 0 returneras.
* Refererar parametern till ett objekt vars användarnamn ska sorteras före det anropande objektets användarnamn ska ett heltal större än 0 returneras.
* Refererar parametern till ett objekt ett objekt vars användarnamn är samma som det anropande objektets användarnamn ska heltalet 0 returneras.

#### Metoden ToString

Metoden ska returnera en sträng som beskriver en anställd. Strängen ska vara väl formaterad och innehålla väl valda egenskaper.

### Klassen Repository

En instans av klassen Repository används för att hantera persistent lagrade anställda och underlag till användarkonton.



Figur 6. Klassen Repository

#### Fältet \_sourcePath

Privat fält av typen string innehållande sökvägen till den fil med anställda en instans av Repository arbetar mot.

#### Egenskapen SourcePath

Publik egenskap av typen string som kapslar in fältet \_sourcePath. set-metoden ska validera sökvägen så att den inte refererar till null, är tom eller bara innehåller vita tecken (”*white spaces*”) och att filen existerar genom att använda File.Exists().

#### Egenskapen DestinationPath

Autoimplementerad privat egenskap av typen string innehållande sökvägen till den fil som det genererade underlaget till användarkonton ska skrivas. Egenskapen tilldelas ett värde av konstruktorn.

#### Konstruktorn

Konstruktorn ska initiera fältet \_sourcePath, via egenskapen SourcePath, så att det instansierade objektet innehåller en giltig sökväg till en fil med anställda. Vidare ska egenkapen DestinationPath tilldelas en sökväg till den fil som underlaget för användarkonton ska skrivas till. Sökvägen ska vara densamma som källfilen med tillägget att .usr ska läggas till innan filändelsen. Har källfilen namnet Employees.csv ska destinationsfilens namn vara Employees.usr.csv.

#### Metoden Load

Den publika metoden Load() ska läsa in textfilen med anställda och tolka den för att skapa en lista med referenser till Employee-objekt som returneras.

Under rubriken ’Format på textfil med anställda’ finns information om textfilen format. Under rubriken ’’ finns en algoritm som kan användas för att läsa in och tolka textfilen.

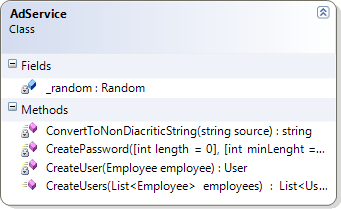
Uppstår fel under inläsningen eller tolkningen, t.ex. på grund av att textfilen inte är korrekt formaterad, ska metoden kasta ett undantag.

#### Metoden Save

Den publika metoden Save() ska spara det underlag till användarkonton som skickas med som argument vid anrop av metoden. Underlaget med användarkonton ska spara enligt det format som krävs för att befintligt skript ska fungera.

### Klassen AdService

Klassen AdService används till att generera underlag till anställdas användarkonton.



Figur 7. Klassen AdService där metoderna CreateUser och CreateUsers saknas och måste implementeras.

#### Metoden ConvertToNonDiacriticString

Den privata metoden ConvertToNonDiacriticString() är redan implementerad och används för att ta bort diakritiska tecken, d.v.s. små skrivtecken som läggs till en bokstav (över, under eller ovanpå).

#### Metoden CreatePassword

Den privata metoden CreatePassword() är redan implementerad och används för att generera ett unikt lösenord som uppfyller (nästan) ställda krav på komplexitet (bestå av minst sex tecken, innehållande en kombination av minst tre av följande tecken: VERSALER, gemener, siffror och symboler, inte innehålla användarnamnet (inte uppfyllt med 100 % säkerhet!)).

#### Metoden CreateUser

Den privata metoden CreateUser() ska skapa, initiera och returnera en referens till ett User-objekt baserat på argumentet, en referens till ett Employee-objekt, som skickas till metoden. Användarnamnet ska bestå av de två första bokstäverna i för- respektive efternamn. *OBS! Användarnamnet behöver inte göras unikt i denna metod, det görs enklast i metoden CreateUsers*.

#### Metoden CreateUsers

Den publika metoden CreateUsers() ska utifrån en lista med referenser till Employee-objekt skapa en lista, sorterad på användarnamn, med referenser till User-objekt.

Metoden ska säkerställa att en användares användarnamn är unikt. Det får inte vara någon kollision mellan befintliga användarnamn eller nya användarnamn. Befintliga användarnamn kan slås upp med hjälp av klassen FakeActiveDirectory.

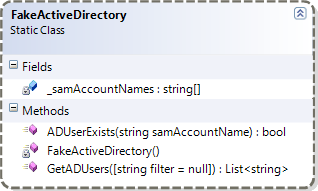
Existerar redan ett användarnamn ska användarnamnet kompletteras med ett suffix i form av ett tal. Finns redan användarnamnet albr ska användarnamnet albr2 skapas. Finns användarnamnet misu9 ska användarnamnet misu10 skapas.

Ett användarnamn består garanterat av minst fyra tecken varför metoden String.Substring() kan användas för att ta fram ett eventuellt suffix. Om variabeln userName har värdet ”misu9” returnerar userName.Substring(4) strängen ”9” som enkelt kan göras om till ett heltal vars värde sedan ökas med ett för att slutligen läggas till användarnamnet som då blir ”misu10”.

### Klassen FakeActiveDiretory

Klassen FakeActiveDiretory simulerar ett ”*active directory*” med ett antal användarnamn. Klassen innehåller två statiska metoder som kan användas för att slå upp befintliga användarnamn (GetADUsers) eller undersöka om ett användarnamn redan existerar (ADUserExists).

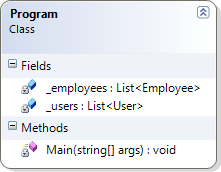
**Klassen får under inga omständigheter ändras på något sätt.**



Figur 8. Klassen FakeActiveDiretory som är fullständigt implementerad.

### Klassen Program

Klassen Program ska använda instanser av lämpliga objekt för att läsa in anställda och spara underlag för användakonton.



Figur 9. Samtliga medlemmar i klassen Program måste vara statiska.

#### Fältet \_employees

Privat statiskt fält av typen List<Employee> innehållande referenser till objekt med information om anställda.

#### Fältet \_users

Privat statiskt fält av typen List<User> innehållande referenser till objekt med underlag till användarkonton.

#### Metoden Main

Metoden Main ska instansiera objekt av typerna Repository och AdService.

Repository-objektet ska användas för att läsa in information om anställda från den textfil som skickas som argument med in till applikationen. AdService-objektet ska sedan användas till att skapa underlag för användarkonton. Avslutningsvis används Repository-objektet igen, denna gång för att skriva underlaget för användarkontona till en textfil.

Eventuella fel ska fångas av metoden och felmeddelande presenteras.

## Krav

Samtliga krav som ställs under rubrikerna ovan ska vara uppfyllda.

## Läsvärt

* Strukturer
  + Essential C# 4.0, 332-337.
  + <http://msdn.microsoft.com/en-us/library/ah19swz4.aspx>
* Sortera med OrderBy()
  + Essential C# 4.0, 566-568.
  + <http://msdn.microsoft.com/en-us/library/6sh2ey19.aspx>
* Klassen List<T>
  + Essential C# 4.0, 617-621.
  + <http://msdn.microsoft.com/en-us/library/6sh2ey19.aspx>
* Läsa och skriva till textfiler
  + StreamReader, <http://msdn.microsoft.com/en-us/library/6aetdk20.aspx>.
  + StreamWriter, <http://msdn.microsoft.com/en-us/library/3ssew6tk.aspx>
  + <http://voyager.lnu.se/tekinet/kurser/dtt/1dv402/files/Hur%20hanterar%20jag%20textfiler.pdf>
* Klassen Path
  + <http://msdn.microsoft.com/en-us/library/3bdzys9w.aspx>