**TRF**

**Tigerälskarnas Riksförbund**

Rapport

**Programmering 2 C#**

**NTI-Skolan**

**Johan Kämpe**<https://www.linkedin.com/in/johankampe><https://github.com/GoblinDynamiteer>

**2017-07-17**

Innehåll

[***1 Inledning*** 3](#_Toc488052648)

[**1.1 Syfte** 3](#_Toc488052649)

[**1.2 Noteringar** 3](#_Toc488052650)

[**1.3 Länkar** 3](#_Toc488052651)

[***2 Genomförande och resultat*** 4](#_Toc488052652)

[**2.1 Använd programvara och litteratur** 4](#_Toc488052653)

[**2.2 Avgränsningar och krav** 4](#_Toc488052654)

[2.2.1 Grundläggande funktionalitet 4](#_Toc488052655)

[2.2.2 Utökad funktionalitet 4](#_Toc488052656)

[**2.3 Metod** 5](#_Toc488052657)

[**2.4 Programmets funktion** 5](#_Toc488052658)

[**2.5 Flödesschema** 9](#_Toc488052659)

[**2.6 Pseudokod** 10](#_Toc488052660)

[**2.7 Skillnader från uppgift 3** 11](#_Toc488052661)

[**2.8 Programmets kod** 11](#_Toc488052662)

[2.8.1 Klassen Logbook 11](#_Toc488052663)

[2.8.2 Klassen Menu 12](#_Toc488052664)

[***3 Diskussion och slutsats*** 13](#_Toc488052665)

[**3.1 Förslag på förbättringar** 13](#_Toc488052666)

[**3.2 Val av algoritmer, variabler och datastrukturer** 15](#_Toc488052667)

[**3.3 Utvärdering av insats** 16](#_Toc488052668)

# 1 Inledning

## Syfte

Syftet med uppgiften är att skapa ett program i programspråket C#, som underlag finns en projektspecifikation med vissa krav som måste uppfyllas.

Utöver dessa krav får programmet skapas med fria händer.

## Noteringar

Skärmklipp i rapporten visar ibland en äldre version av programmet, vissa skillnader kan förekomma i utseende och text i det inlämnade programmet.

## 1.3 Lösenord

Programmet startar med ett loginfönster, där ett lösenord måste skrivas in för att komma vidare.



Figur 1 loginfönster

Lösenordet sparas i en fil kallad *login.pwd*. Om denna fil av någon anledning saknas i den katalog där programmets exekverbara fil ligger används ett standardlösenord.

|  |  |
| --- | --- |
| Lösenord för att logga in i programmet | |
| TigrarÄger123! | Om filen login.pwd finns tillgänglig för programmet |
| 123 | Standardlösenord om login.pwd inte används |

Användaren har möjlighet att själv byta lösenord i programmet, om så önskas.

## Länkar

Uppgiften på GitHub  
<https://github.com/GoblinDynamiteer/P2CS/tree/master/course_exercises/project>

Ikoner som används i programmet  
<https://www.iconfinder.com/icons/381599/error_icon>  
<https://www.iconfinder.com/icons/285654/cat_icon>

# 2 Genomförande och resultat

## 2.1 Använd programvara och litteratur

Programvara

* Microsoft Visual Studio Community 2017 v4.7.02045
* Microsoft Word 2016 v1701
* Draw.io: <https://www.draw.io>
* Trello, planeringsverktyg: <https://trello.com>

Litteratur

* Alishenas, T *Programmering 2 med C#* (ISBN 9789197420433)

## 2.2 Avgränsningar och krav

### 2.2.1 Kravspecifikation

Medlemmarna i TRF består av människor som på något sätt skaffat sig en tiger som husdjur. Ditt jobb här blir nu att skapa ett program som håller reda på namn och adress till medlemmarna och namn på tigern/tigrarna.

**Du får helt fria händer, bortsett från nedanstående punkter:**

* Programmet **ska** innehålla en databas, och det ska givetvis då gå att lägga till/ta bort medlemmar.
* Det **bör** någonstans finnas en länk till Tiger på Wikipedia, klickar man på den så ska webbsidan visas (<http://sv.wikipedia.org/wiki/Tiger>)

**Tillägg för högre betyg:**

* När man startar programmet så ska man komma till någon sorts inloggningsruta, och lösenord osv. ska i så fall sparas i en textfil på hårddisken (gärna krypterad).
* Kom ihåg att nämna inloggningsuppgifterna i rapporten

**Mer att tänka på:**

* Programmet bedöms till mycket stor del efter hur du löst allt (inklusive upplägg av databasen).
* Använd minst två klasser och försök att dela upp koden med metoder osv.
* Klassdiagram ***ska*** finnas med.
* Någon form av felhantering för att undvika körfel osv. ***bör*** finnas med.

### 2.2.2 Redovisning

**Programmet**

* Programmet ska fungera och uppfylla kravspecifikationen.
* Redovisa programmet genom att **skicka in källkodsfilerna** via din elevsida (komprimera ihop hela projektmappen).
* Programmet **ska** vara försett med **utförliga kommentarer** som förklarar de olika delarna i programmet.
* Programmet **ska** vara utan stilfel, dvs. med korrekta indrag och radbrytningar.
* För högre betyg **bör** programmet **utökas** med annan funktionalitet, utöver det som anges i beskrivningen, så länge detta inte strider mot den grundläggande beskrivningen, och uppfyller kraven.

**Dokumentation**

* **Projekt-rapport** ska skickas in tillsammans med källkoden en där du **beskriver** hur du gick till väga för att skapa programmet. Vilka avvägningar du gjort i programupplägget och motiveringar till dessa. Du **bör** också ge exempel på eventuella förbättringar.
* Uppgiften ska planeras med hjälp av pseudokod och/eller flödesschema som du sedan skickar in tillsammans med programmet.

## 2.3 Metod

### 2.3.1 Arbete med Visual Studio

Programmets källkod skrevs uteslutande i utvecklingsmiljön *Visual Studio.* Som projektmall användes *Windows Forms App (.NET Framework).*

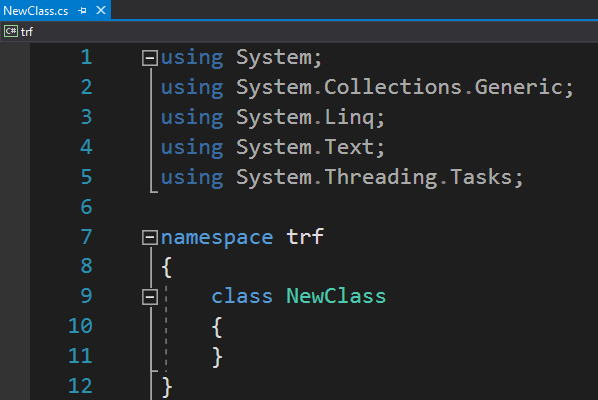
I Visual Studio finns ett hjälpverktyg kallat *IntelliSense,* som underlättar vid kodskrivning.

Visual Studio har också stöd för debugging med breakpoints, vilket användes för att felsöka programmet.



Figur 2 Kod i Visual Studio, debugläge med break point

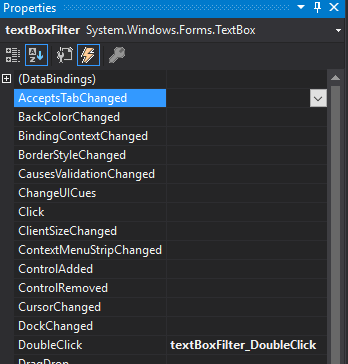
Egna klasser lades till i projektet genom att välja *”Project -> Add Class…”* i Visual Studios meny. Visual studio skapar då en ny cs-fil i projektet, med färdiggenererad kod för att skapa en ny klass.



Figur 3 Ny klassfil skapad i Visual Studio

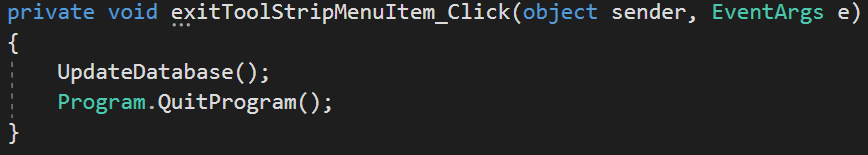
På liknande sätt skapades nya Windows Forms till projektet, med *”Project -> Add Windows Form…”*

De flesta event-metoder som finns i projektet skapades genom att dubbelklicka på den kontroll som de tillhör, i Visual Studios Form Design-läge, eller genom att dubbelklicka på det tomma utrymmet till höger om det önskade eventet i kontrollens egenskaper.



Figur 4 Events för kontrollen textBoxFilter

Namnen som Visual Studio automatiskt genererar till metoderna behölls i de flesta fall.

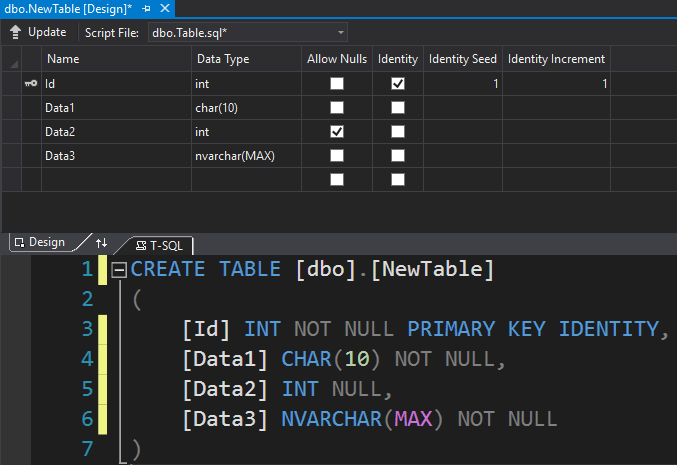


Figur 5 Automatiskt genererat metodnamn för event till kontroll

Databasen som används i programmet skapades och hanterades också i Visual Studio.

En ny databas läggs till i ett projekt med *"Project -> Add New Item -> Service-based Database"*  
Kapitel 5 i läroboken *Programmering 2 med C#* användes i stor grad som hjälpmedel vid arbetet med databasen.

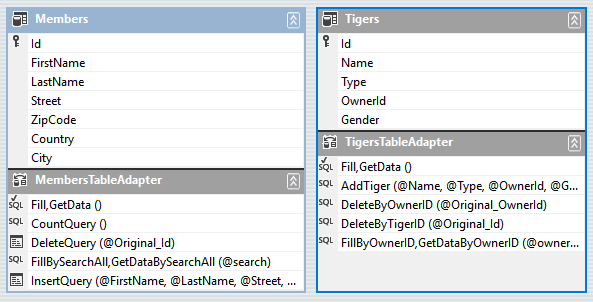
Databasens tabeller skapades i Visual Studio med hjälp av dess *”Table Designer”.*



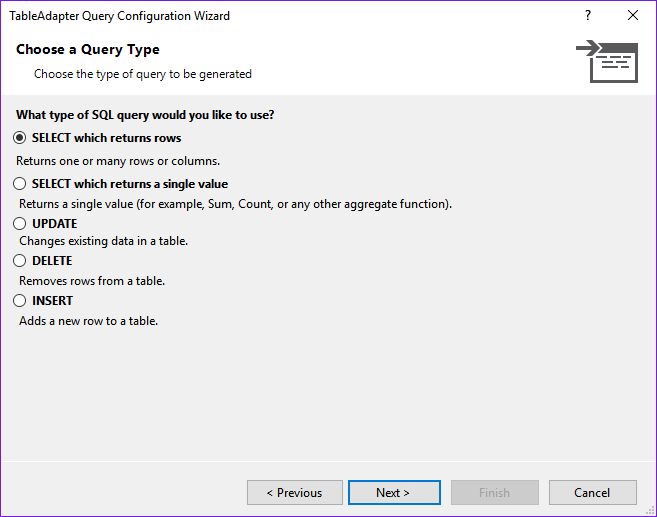
Figur 6 Visual Studios Table Designer

Visual Studio använder sig av så kallade ”Table Adapters” för att sköta kommunikation mellan databaser och projektets kod. Till dessa adapters är det möjligt att skapa SQL-kommandon, som sparas som metoder till adapter-klassen.

Till projektet skapades flertalet SQL-kommandon (query), för att exempelvis söka bland medlemmar i TRF eller lägga till eller ta bort medlemmar. SQL-kommandona kan skapas med hjälp av guider i programmet.



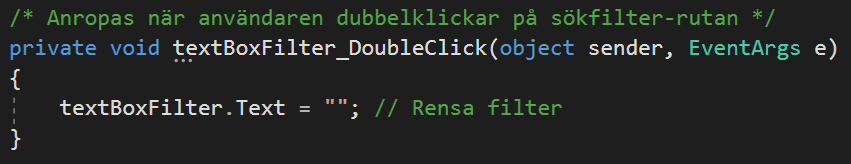
Figur 7 Table Adapters med SQL-queries



Figur 8 SQL-guide i Visual Studio

### 2.3.2 Kommentering av kod

Programmets kod är kommenterad, där koden inte är automatiskt genererad av Visual Studio. Flerradskommentarer ( /\* KOMMENTAR \*/ ) beskriver den kod som finns på raden under medans enradskommentarer ( // KOMMENTAR ) beskriver den kod som finns till vänster om kommentaren.

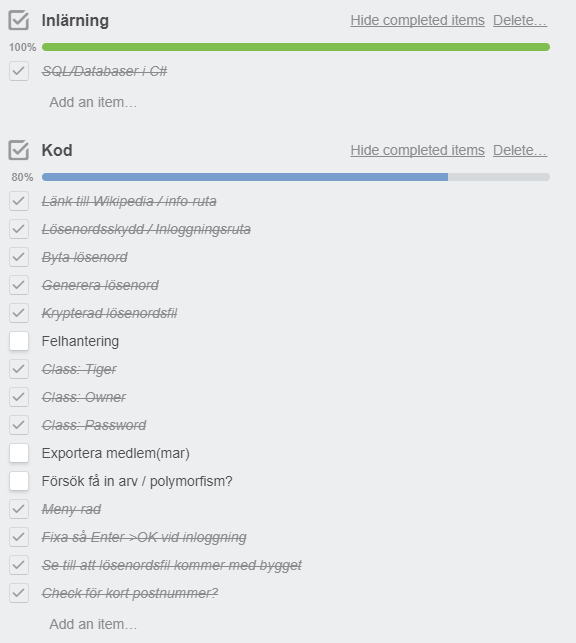


Figur 9 Kommentarer i källkoden

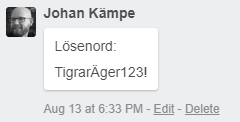
## 2.4 Planering

Projektet planerades genom att skapa ett flödesschema med webverktyget draw.io, endast programmets grundläggande funktion, samt inloggningsruta lades till.

Det webbaserade planeringsverktyget Trello användes som checklista och som en sorts ”dagbok” för att skriva ned tankar och kommentarer kring utvecklingen av programmet.

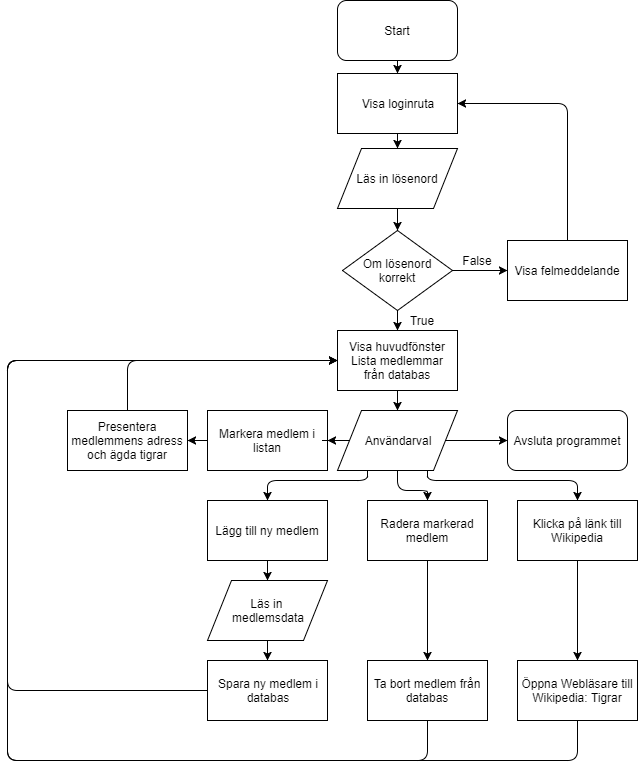


Figur 10 Checklistor i Trello



Figur 11 Kommentar i Trello

### 2.4.1 Flödesschema



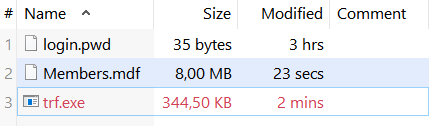
Figur 12 Flödesschema

## 2.5 Programmets funktion

### 2.5.1 Inlogging och filer

Programmet startas med den exekverbara filen *trf.exe*. För att programmet ska fungera korrekt behövs också databasfilen *Members.mdf* i samma katalog.

En tredje fil, *login.pwd* tillhör också programmet, denna fil innehåller ett sparat lösenord för att logga in. Denna fil är dock inte nödvändig för programmets funktion.



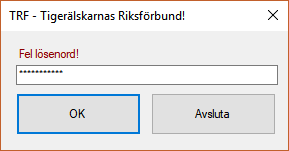
Figur 13 Programmets filer

Vid uppstart visas en inloggningsruta, där användaren uppmanas att skriva in ett lösenord för att fortsätta.



Figur 14 Inloggningsruta

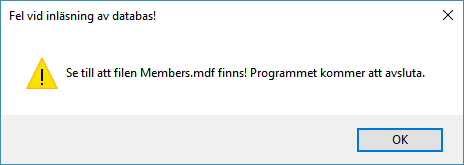
Om ett felaktigt lösenord skrivs in och användaren trycker på *OK*-knappen visas ett felmeddelande. Programmet avslutas om användaren trycker på ”krysset” eller *Avsluta*-knappen.



Figur 15 Felmeddelande om fel lösenord

Om filen *login.pwd* fanns vid programmets uppstart är lösenordet **TigrarÄger123!**, annars är lösenordet **123**. Om användaren har valt ett annat lösenord vid ett tidigare tillfälle så är det det aktuella lösenordet.

Om databasfilen *Members.mdf* saknas kommer ett felmeddelande att visas efter lyckad inloggning, och programmet avslutas.

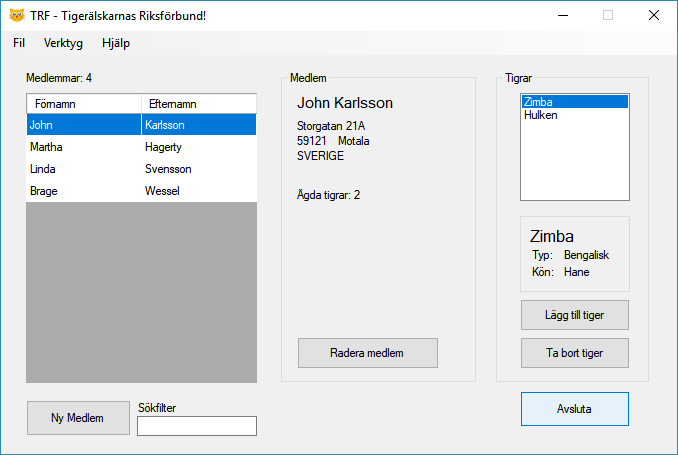


Figur 16 Felmeddelande om databas

Filerna *Members.mdf* och *login.pwd* kopieras alltid automatiskt av Visual Studio till det kompilerade programmets katalog vid varje kompilering. Det innebär att ändringar (databasändringar eller lösenordbyte) som har utförts vid körning av programmet via Visual Studio kommer att återställas vid varje ny körning via Visual Studio.

Om databasfilen finns, och den laddas in av programmet korrekt, så presenteras programmets huvudfönster.

### 2.5.2 Beskrivning av programmets huvudfönster



11

10

9

8

7

6

5

4

3

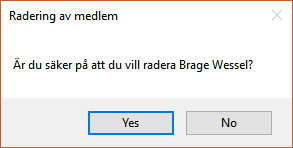
2

1

Figur 17 Huvudfönster med beskrivning

1. Medlemslista, användaren klickar på ett namn för att välja en annan medlem. Listan kan sorteras genom att klicka på rubrikerna ”Förnamn” eller ”Efternamn”.
2. Lägger till en ny medlem, ett nytt fönster öppnas.
3. Information om den markerade medlemmen.
4. Radera den listade medlemmen. En dialogruta visas för att bekräfta.

10

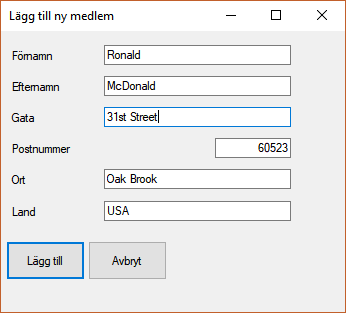
1. Den markerade medlemmens tigrar.

Figur 18 Radering av medlem

1. Information om den markerade tigern.
2. Lägg till en ny tiger till den markerade medlemmen. Ett nytt fönster öppnas.
3. Ta bort markerad tiger från den markerade medlemmen. En dialogrutas visas för att bekräfta.
4. Filtrera medlemslistan med sökterm.
5. Menyrad.
6. Avsluta programmet.

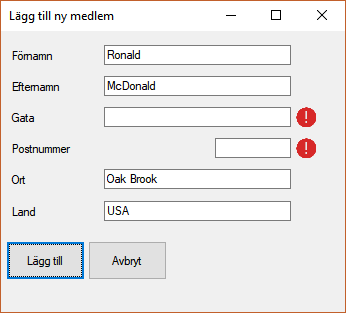
### 2.5.3 Lägg till en ny medlem

När användaren trycker på knappen *Ny Medlem* öppnas ett nytt fönster där användaren ombeds fylla i den nya medlemmens namn och adress.



Figur 19 Fönstret Lägg till ny medlem

Om formuläret inte är korrekt ifyllt när användaren trycker på *Lägg till*-knappen visas felikoner vid de felaktiga inmatningarna.



Figur 20 Felaktig inmatning av ny medlem

Om formuläret är korrekt ifyllt läggs den nya medlemmen till i databasen. Användaren kan också välja att avbryta inmatningen genom att trycka på *Avbryt*-knappen.

### 2.5.4 Lägg till en ny tiger.

### 2.5.5 Sökfilter

### 2.5.6 Meny Fil

### 2.5.7 Meny Verktyg: Ändra lösenord

### 2.5.8 Meny Hjälp

## 2.8 Programmets kod

Källkoden beskrivs enbart övergripande i denna rapport, i den inlämnade källkodsfilen finns ytterligare kommentarer som beskriver koden.

Kod som är automatiskt genererad av Visual Studio beskrivs eller kommenteras ej.

### 2.8.1 Klassen Logbook

Klassen ”class Logbook” innehåller en lista av string-arrayer ”List<string[]> entries”, som används för att hålla loggbokens inlägg. Varje inlägg består av en array med fyra string-variabler; Titel, ID-nummer, Text-innehåll, och datum/tid.

Metoder i klassen Logbook

Flera metoder i klassen Logbook finns som överlagrade, en variant där parametrar skickas till metoden direkt och en där användaren ombeds skriva in data manuellt.

|  |  |
| --- | --- |
| Metod | Beskrivning |
| Logbook | Konstruktor för klassen. Laddar in sparad loggboksdata från fil om den existerar. Konstruktorn kan användas för att skapa generera testinlägg. |
| Count | Ger antal inlägg i loggboken. |
| ClearSearchHits | Nollställer sökresultat. |
| Search | Söker i logginlägg efter en angiven söksträng, finns som överlagrad. Metoden använder *linjär sökning*. |
| DisplaySearchHits | Listar titlarna på inlägg från det senaste sökresultatet. |
| DisplayAllTitles | Listar titlarna på samtliga inlägg i loggboken |
| DisplayTitle | Listar titeln på angivet inlägg, inläggets ID skickas som parameter. |
| AddEntry | Lägger till ett inlägg till loggboken, finns som överlagrad. |
| DeleteEntry | Raderar ett inlägg från loggboken. |
| DisplayEntry | Visar innehållet i ett inlägg, finns som överlagrad. Visar också en lista med alternativ för det aktuella inlägget: exportera, radera, redigera titel, redigera text. |
| EditTitle | Redigerar titel på ett inlägg. |
| EditContent | Redigerar texten i ett inlägg. |
| ExportEntry | Exporterar ett inlägg till en textfil. |
| Save | Sparar loggboken till en datafil. |
| Load | Laddar loggboken från en datafil. |
| UpdateEntry | Ändrar titel och/eller text för ett inlägg. Anropas av metoderna EditTitle och EditContent. |
| Sort | Sorterar loggboksinlägg. Finns som överlagrad. Metoden använder *insertion sort.* |
| SortSwap | Hjälpmetod för Sort, byter indexplats på två loggboksinlägg. |
| FindEntryIndex | Letar upp indexplats för ett inlägg, tar ID som parameter. |
| AddSamples | Lägger till några förskrivna logginägg till loggboken, för testning. |

Variabler och datastrukturer i klassen Logbook:

|  |  |  |
| --- | --- | --- |
| Namn | Typ | Beskrivning |
| entries | List<String[]> | Håller loggboksinlägg med titel, text, ID-nummer och datum. |
| searchHits | int[] | Array som håller sökträffar, som ID-nummer. |
| lastSearchString | string | Senaste söksträngen som användes för sökning |
| dataFileName | string | Filnamn för datafil som sparar loggboken mellan programkörningar. |
| id | int | Används för att sätta ID-nummer på nya logginlägg. |
| SearchData | enum | Värden för sökningsmetoder, tom och max antal sökträffar. |
| sortMenu | string[] | Text för den meny som visas vid val av sortering. |
| displayEntryMenu | string[] | Text för de alternativ som kan göras när ett inlägg visas. |
| SortBy | enum | Sorteringsalternativ. |
| EntryData | enum | Indexnummer för text, ID-nummer, datum och titel i char-arrayen i listan *entries*. |
| errorMsg | string[] | Felmeddelanden. |
| ErrorId | enum | Index för felmeddelanden. |

### 2.8.2 Klassen Menu

Klassen ”class Menu” innehåller metoder för att visa menyer och diverse text, samt funktionalitet för användarinmatning.

Metoder i klassen Menu:

|  |  |
| --- | --- |
| Metod | Beskrivning |
| DisplayMenu | Visar programmets huvudmeny, ger i retur användarens inmatade menyval. |
| DisplayTitle | Genererar titlar för programmets olika menyer. Titeltexten anges som parameter. Texten centreras. |
| DisplayLine | Ritar ut en linje till konsollfönstret. |
| Wait | Uppmanar användaren att trycka på en valfri tangent för att fortsätta programmet. Meddelande kan ges som parameter. |
| Confirm | Ger i retur ett sanningsvärde, beroende på om användaren trycker in en viss tangent. Exempel: *”Tryck på ’J’ för att ta bort filen”.* |
| Error | Visar ett felmeddelande, meddelandet anges som parameter. |

Variabler och datastrukturer i klassen Menu:

|  |  |  |
| --- | --- | --- |
| Namn | Typ | Beskrivning |
| MenuItem | enum | För navigering i huvudmenyn. |
| lineLen | int | Längd för linjer, används i DisplayLine och DisplayTitle |

# Diskussion och slutsats

## 3.1 Förslag på förbättringar

ETC