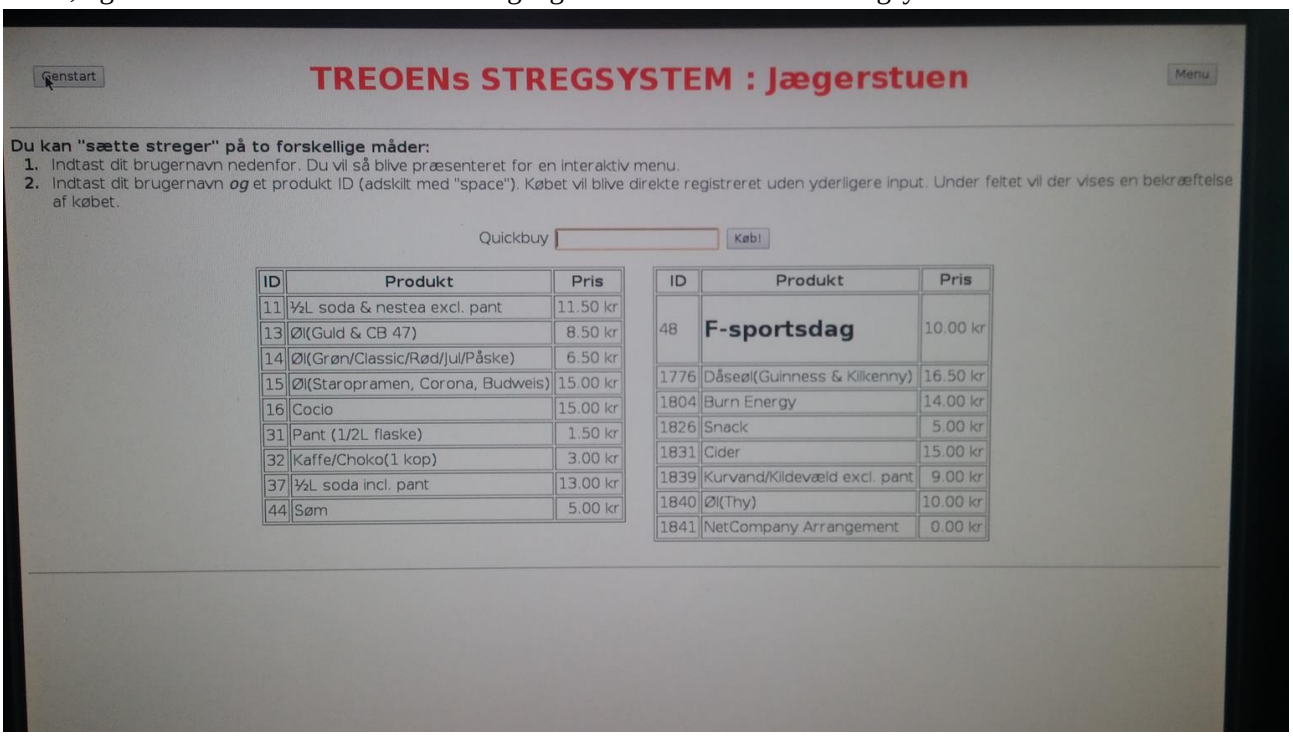
Eksamensopgave i OOP forår 2017

Introduktion

F-klubben (fklub.dk) er en forening for alle studerende og ansatte indenfor IT og naturvidenskabelige uddannelser på Aalborg Universitet. Formålet med foreningen er at opretholde det sociale liv - også på tværs af studieretninger. F-klubben sørger bl.a. for at der findes friske forsyninger af drikkevarer, og bliver holdt arrangementer. Man betaler for disse ting, ved at sætte en streg, når man tager en vare i køleskabet, eller tilmelder sig et arrangement. I starten var det på papir, men i moderne tider, er **stregsystemet** blevet digitaliseret. Den elektroniske version af stregsystemet blev påbegyndt i oktober 1996, og er siden blevet omskrevet en del gange. Et seneste billede af stregsystemet kan ses herunder:



På hovedskærmen ses en liste af produkter, der kan købes i stregsystemet. Dette kan variere bl.a. ved sæsoner. Der er to måder at købe på; man kan klikke sig frem med musen, og og man kan bruge quickbuy. Der bliver dog stort set kun brugt quickbuy. Quickbuy er et simpelt kommandointerface, der også er beskrevet på hovedskærmen. Vil man købe en vare, f.eks. en ”sodavand eller nestea (excl pant)” til 11,50 kr, skriver man sit brugernavn, efterfulgt af 11, og så bliver der trukket 11,50 kr på ens konto, og man må tage en sodavand.
Der er altid snak om at omskrive stregsystemet til en mere avanceret og mere moderne udgave, bl.a. med den famøse multi-quickbuy. Det er nu blevet jeres tur til at lave en udgave af stregsystemet!

Eventuelle uklarheder i opgaveformuleringen forventes afklaret som en del af opgaveløsningen. I bestemmer selv hvilket sprog I skriver koden på, men undgå venligst æøå og lign. i filnavne. Jeg har valgt at bruge engelske navne for at undgå danske tegn.

Opgaven er delt op i flere dele:

1. **Stregsystemets kerne**
2. **Brugergrænsefalde**
3. **Kontrol af kerne og brugergrænseflade**
4. **Sammensætning**

For at lette opgaven, kan du finde et skabelonprojekt sammen med denne opgaveformulering.

Læs grundigt hele opgaven igennem inden du begynder.

Krav til aflevering:
(følges disse ikke, kan din aflevering blive afvist)

1. Øverst (linie 1) i den .cs-fil der indeholder din Main, skal der i en kommentar stå dit studienummer efterfulgt af dit fulde navn på formen:
   //20121234_Peter_Aage_Nielsen
2. Programmet afleveres som **Visual Studio Projekt** online på moodle, pakket i en **ZIP fil**. Denne fil navngives med navn og studienummer på formen:
   Studienummer_fuldt_navn.zip
   Eksempel:
   20121234_Peter_Aage_Nielsen.zip
   - **Ellers ved jeg ikke hvem du er, og kan ikke registrere din aflevering!**
3. Det udviklede program skal være objekt-orienteret, og skal skrives i C#.
4. Det udviklede program skal være **demonstrerbart**, dvs. det skal kunne afvikles til eksamen.
5. Hver offentlig/internal type skal placeres i en .cs fil med samme navn som typen:
   - User placeres i User.cs etc.
6. Med det udviklede program skal der følge en kort beskrivelse af programmet (ca. én side). Denne skal afleveres som **PDF**.
   Denne beskrivelse skal omfatte:
   - Status af programmet, inklusiv en beskrivelse af eventuelle antagelser, begrænsninger, udvidelser, eller variationer i forhold til opgaveformuleringen.
   - Angivelse af eventuelle kilder og hjælp (dog ikke kursusbogen eller kursusslides)

Del 1. Stregsystemets kerne

Stregsystemet skal administrere **brugere**, **produkter**, og **transaktioner** mellem disse.

User

En **bruger** har følgende egenskaber:

- ID
  - et unikt tal-id der identificerer brugeren. Jo lavere tallet er, jo før blev man indmeldt i F-klubben. Dette tal er unikt blandt brugere.
- Firstname
  - Brugerens fornavn(e). Dette må aldrig være **null**.
- Lastname
  - Brugerens efternavn. Dette må aldrig være **null**.
- Username
  - Brugerens brugernavn. Dette bliver brugt når der bliver købt et produkt (sat en streg). Det er brugernavnet alene, der identificerer brugeren i stregsystemet! Brugernavnet må indeholde 0-9, små bogstaver og underscore: [0-9], [a-z], og '_' [1]
- Email
  - Brugerens email-adresse. Dette skal altid være en gyldig email-adresse efter følgende krav.
  - En email adresse består af **local-part@domain,** og kravene til de to dele er:
    - **local-part** må bestå af **a-z, A-Z,** og tallene **0-9** samt tegnene punktum, underscore og bindestreg ('.', '_' og '-')
    - **domain** må indeholde **a-z, A-Z,** og tallene **0-9** samt punktum og bindestreg. **domain** må ikke starte eller slutte med bindestreg/punktum. **domain** skal indeholde mindst et punktum.
      - Et eksempel på en gyldig email-adresse er: eksempel@domain.dk
      - Et eksempel på en ugyldig email-adresse er: eksempel(2)@-mit_domain.dk
- Balance
  - Brugerens saldo. Man indbetaler penge på sin konto, og herefter kan man købe varer i systemet. Når brugerens saldo går under 50 kroner skal brugeren advares, og ved yderligere køb bliver der vist en advarselsbesked. Introducér et delgate til denne type events
    - `UserBalanceNotification(User user, decimal balance)`

- User skal have en tostring bestående af:
  - **Fornavn(e) Efternavn (Email)**
- Klassen skal også implementere en fornuftig Equals-metode samt GetHashCode.
- Klassen skal implementere IComparable, og sorteres på ID.
- Klassen skal implementere en fornuftig konstructor

Product

Et **produkt** har følgende egenskaber:

- ID
  - ID er et unikt tal-id for produkter. Dette bruges når man skal købe et produkt. Produktets ID må ikke være under 1. Dette ID bruges senere til at identificere produkter unikt, bl.a. ved køb.

[1] Det bør ikke være nødvendigt at bruge regulære udtryk (regex) til nogle af disse valideringsfunktioner – heller ikke email! Man kan sagtens skrive en lille metode til at validere.

- Name
  - Beskrivelse af produkt. Et eksempel kunne være ”½L vand & kildevand excl. pant". Navn må aldrig være **null**.
- Price
  - Prisen på et produkt. Denne vil ofte blive justeret.
- Active
  - Et flag der afgør om et produkt er aktivt eller ej. Der findes produkter der ikke sælges mere, men stadig findes i databasen (det kan være de ikke kan skaffes, eller af anden årsag ikke er i stregsystemet mere).
- CanBeBoughtOnCredit
  - Et flag der afgør om et produkt kan købes, selvom brugeren ikke har penge nok på sin konto. Der findes produkter, f.eks. fester, som folk kan finde på at købe i sidste øjeblik. For ikke at udelukke impulsive folk, med for få penge på sin konto, fra arrangementer, kan nogle produkter købes på kredit. Som udgangspunkt skal der altid være penge nok på ens konto, hvis man vil købe et produkt.
- ToString kan med fordel returnere produktets ID, Navn og Pris, som ses på billedet.
- Klassen skal have en fornuftig konstructor

SeasonalProduct

I denne nye version af stregsystemet vil vi introducere en ny type produkt, **SeasonalProduct**, for at reducere manuelt arbejde**.** Denne type produkter har følgende egenskaber:

- ID
  - ID er et unikt tal-id for produkter. Dette bruges når man skal købe et produkt. Produkt-ID må ikke være under 1 og skal selvfølgelig være unikt for hvert produkt.
- Name
  - Beskrivelse af produkt. Et eksempel kunne være ”½L vand & kildevand excl. pant". Navn må aldrig være null.
- Price
  - Prisen på et produkt.
- CanBeBoughtOnCredit
  - Et flag der afgør om et produkt kan købes, selvom brugeren ikke har penge nok på sin konto.
- SeasonStartDate
  - Specificerer tidspunktet sæsonen starter på.
- SeasonEndDate
  - Specificerer tidspunktet sæsonen slutter på.
- Active
  - Afgør om et produkt er aktivt eller ej, altså, om vi er inden for sæson eller ej, og om produktet i det hele taget er aktivt i systemet.

Transaction

Transaktioner beskriver køb og indsættelse af penge. Generelt kan man sige følgende om transaktioner:

- ID
  - En transaktion har et unikt ID. Det er vigtigt at kunne skelne transaktioner fra hinanden. Dette ID bestemmer rækkefølgen af transaktioner.
- User
  - Den bruger der er en del af transaktionen. Denne må ikke være null.
- Date
  - Dato for transaktion. Denne dato **skal så vidt muligt** være korrekt!

- Amount
  - Et beløb der beskriver bevægelsen på brugerens konto. Negative beløb hæves og positive beløb indsættes.
- ToString
  - Transaction har en specialiseret ToString metode der beskriver transaktionid, user, beløb og tidspunktet transaktionen blev foretaget.
- Execute
  - Transaction specificerer at der findes en metode kaldet **Execute**. Denne metode udfører transaktionen.
- En Transaction skal have en fornuftig constructor. Det vil bl.a. sige en constructor parameteriseret med User.

Der findes to typer transaktioner: **InsertCashTransaction** og **BuyTransaction**

InsertCashTransaction

InsertCachTransaction bruges når der skal optankes penge på brugerens konto, og har følgende:

- ToString
  - Denne transactionstype har en specialiseret ToString, der fortæller at der er tale om en indbetaling, beløb, bruger, og hvornår indbetalingen blev foretaget og id på transaktionen. (du bestemmer rækkefølge og detaljer)
- Execute
  - Lægger beløbet til brugerens konto.

BuyTransaction

BuyTransaction bruges til at repræsentere køb, og er karakteriseret ved følgende

- Product
  - Det produkt der købes i stregsystemets
- Amount
  - Amount er prisen på produktet på købstidspunktet. Produkter ændrer pris over tid, men ikke med tilbagevirkende kraft!
- ToString
  - Denne transactiontype har en specialiseret ToString der fortæller at der er tale om et køb, beløb, bruger, produkt, hvornår købet blev foretaget, og transaktionens id. (du bestemmer rækkefølge og detaljer)
- Execute
  - Trækker beløbet fra brugerens konto. Der opstår en fejlsituation hvis brugeren ikke har nok penge på sin konto og produktet, der købes, **ikke tillader overtræk**. Til dette formål defineres en exception, se nedenfor.

InsufficientCreditsException er en exception, der skal kastes, når en bruger forsøger, at købe et produkt der ikke er råd til. Denne exception indeholder information om **brugeren** og **produktet,** der forsøges købt, samt en herudfra beskrivende besked om fejlen.

Hvis der forsøges at købe et produkt der ikke er aktivt, skal der også kastes en passende exception.

Stregsystem

Stregsystem indeholder ren logik vedrørende brugere, transaktioner og lignende. Det betyder at stregsystem indeholder informationer om brugere, produkter og transaktioner.

Stregsystemet har følgende skitserede metoder og properties:

- BuyProduct(user, product)
  - udfører den logik der køber et produkt på en brugers konto, ved brug af en transaktion
- AddCreditsToAccount(user, amount)
  - indsætter et beløb på en brugers konto, via en transaktion.
- ExecuteTransaction(transaction)
  - hjælpemetode til at eksekvere transaktioner, og sørge for at de bliver tilføjet til en liste af udførte transaktioner. **Hvis transaktionen altså ikke fejler.**
- GetProductByID(...)
  - Finder og returnerer det unikke produkt i listen over produkter ud fra et produkt-id. Der bliver kastet en brugerdefineret exception hvis produktet ikke eksisterer. Denne exception indeholder information om produkt og beskrivende besked.
- GetUsers(Func<User, bool> predicate)
  - Brugere der overholder predicate
- GetUserByUsername(string username)
  - Finder og returnerer den unikke bruger i brugerlisten ud fra brugernavn. Der bliver kastet en brugerdefineret exception hvis brugeren ikke findes. Denne exception indeholder information om bruger og beskrivende besked.
- GetTransactions(User user, int count)
  - Finder et angivet (via parameter) antal transaktioner relateret til en given **specifik bruger**. Det er de nyeste transaktioner der findes.
- ActiveProducts
  - aktive produkter i stregsystemet på nuværende tidspunkt

Derudover bør der være et event til at indikere at en bruger har faretruende få penge på sin konto(se User)

Generelle krav til stregsystem

Logning: Alle udførte transaktioner skal skrives i en logfil.
Derudover bør der være funktionalitet nok til man kan undgå at få kastet exceptions.

Varekatalog

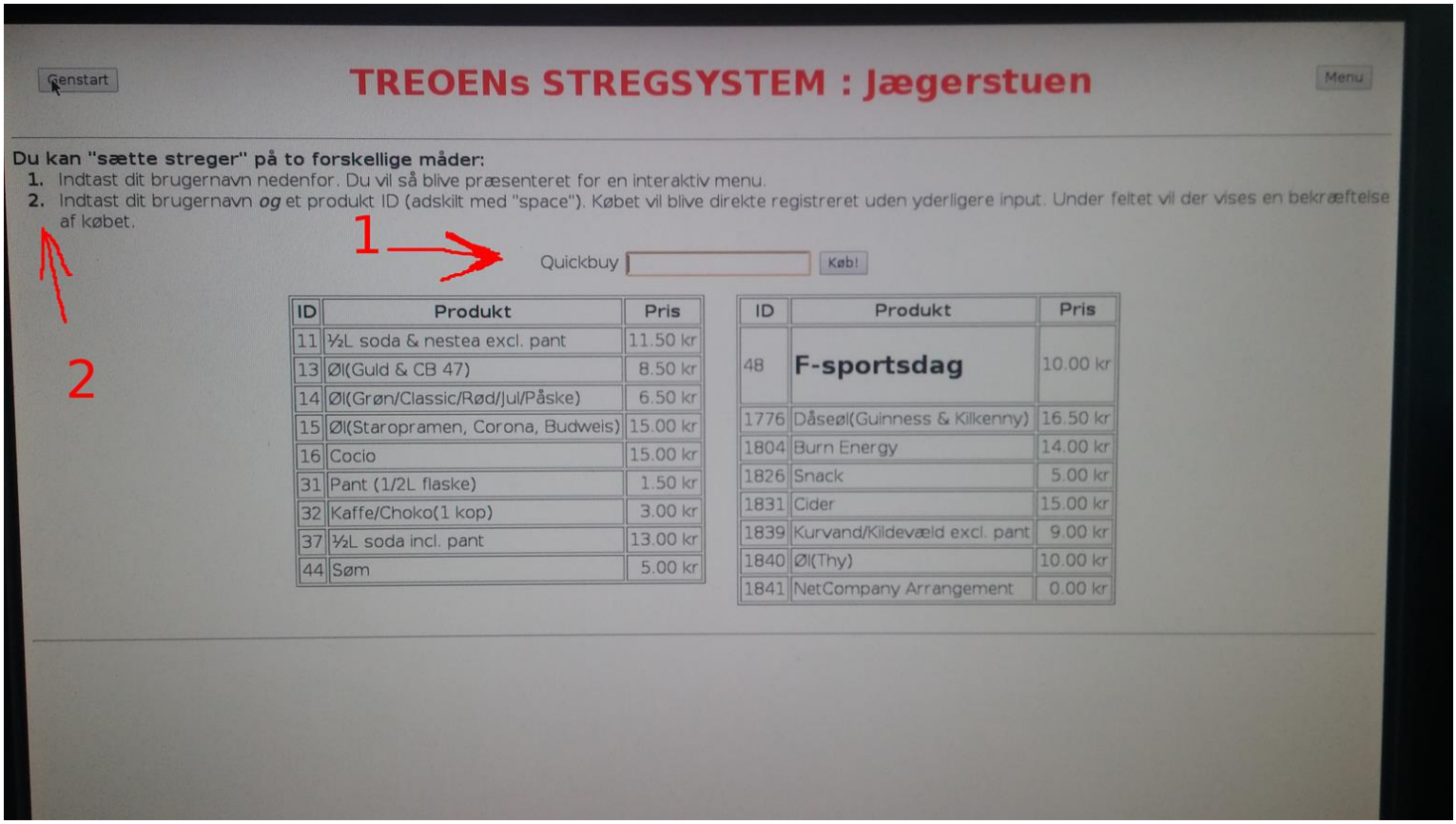
Der er vedlagt en tekstfil med produkter. Der skal skrives en funktion til at indlæse tekstfilen, products.csv, en **semikolon separeret** fil, med produkter. Første linie i filen indeholder beskrivelse af værdier, resten er værdierne. Du kan med fordel ignorere `deactivate_date`.

- Hvis en værdi indeholder mellemrum, er værdien indkapslet i quotes: ”værdi”.
- HTML-tags bør fjernes.
- Hver linie svarer til et produkt
- Værdierne kommer altid i samme rækkefølge
- Der vil **altid** være samme antal semikoloner på en linie!

Denne fil skal indlæses når stregsystemet startes, og aktive produkter skal vises i menuen.
Brugerdatabase: På samme måde som med varekataloget, er der vedlagt en tekstfil med brugere.
Herunder ses et bud på et stregsystemsinterface:

```
public interface IStregsystem
{
    IEnumerable<Product> ActiveProducts { get; }
    InsertCashTransaction AddCreditsToAccount(User user, int amount);
    BuyTransaction BuyProduct(User user, Product product);
    Product GetProductByID(int id);
    IEnumerable<Transaction> GetTransactions(User user, int count);
    User GetUsers(Func<User, bool> predicate);
    User GetUserByUsername(string username);
    event UserBalanceNotification UserBalanceWarning;
}
```

Del 2. Brugergrænseflade



Brugergrænsefladen skal implementeres, som en konsolapplikation. Du har helt frie hænder, men du kan lade dig inspirere af den nuværende brugergrænseflade på figuren ovenfor.
Quickbuy er markeret med 1 og syntaxen for køb er beskrevet i punkt 2.

I denne opgave skal der i UI fokuseres på:

- Præsentation af produkter og priser
- Quick-buy (punkt 2 på figuren), som er den primære købsmetode i stregsystemet.
- Feedback til brugeren i form af tekst.
- Separation af brugergrænseflade

Der skal laves en klasse kaldet **StregsystemCLI**, der repræsenterer et command-line interface til stregsystem-klassen.

StregsystemCLI er karakteriseret ved:

- Start
  - En metode der starter brugergrænsefladen.
  - Når brugergrænsefladen er startet, vil menuen blive vist, og være klar til at modtage quickbuy kommandoer. Mere om kommandoer i 3. del.
  - Brugergrænsefladen skal kun vise aktive produkter.
  - Denne klasse er den eneste i systemet der må skrive noget ud til brugeren!

StregsystemCLI har en reference til stregsystem (gerne et interface som beskrevet tidligere), der bruges til at hente data fra, som skal vises til brugeren.

StregsystemCLI **bør** implementere følgende interface: (det er op til dig hvilke parametre de forskellige metoder tager – der er givet et par bud)

```
public interface IStregsystemUI
{
    void DisplayUserNotFound(string username);
    void DisplayProductNotFound(string product);
    void DisplayUserInfo(User user);
    void DisplayTooManyArgumentsError(string command);
    void DisplayAdminCommandNotFoundMessage(string adminCommand);
    void DisplayUserBuysProduct(BuyTransaction transaction);
    void DisplayUserBuysProduct(int count, BuyTransaction transaction);
    void Close();
    void DisplayInsufficientCash(User user, Product product);
    void DisplayGeneralError(string errorString);
    void Start();
    event StregsystemEvent CommandEntered;
}
```

Disse metoder skal bruges til at vise forskellige informationer til brugeren. Definer selv et passende delegate til `StregsystemEvent.`

F.eks. kunne `DisplayUserNotFound(”testbruger”)` skrive teksten:

```
User [testbruger] not found!
```

Ud til konsollen.

Der må ikke være output til brugeren eller modtages fra brugeren andre steder i programmet end i StregsystemCLI!

Du kan med fordel lade dig inspirere af Start-implementationen beskrevet i Ugeopgave 1.

Del 3. StregsystemController og kobling til brugergrænseflade

StregsystemCommandParser er en klasse der kan oversætte kommandoer i tekststrenge, der skrives af brugeren, til funktionalitet i form af metodekald. StregsystemCommandParser opdaterer brugergrænsefladen og modificerer stregsystemet.
Den er karakteriseret ved:

- Den holder referencer til både Stregsystem og til en `IStregsystemUI`.
  - *På samme måde som `IStregsystemUI` kunne man definere `IStregsystem` som interface til Stregsystem klassen. Derved kan man udskifte Stregsystem, og evt. lettere teste StregsystemCommandParser.*
- Dens primære funktionalitet ligger i metoden ParseCommand(string command).
- En kommando kan have parametre, de er separeret af mellemrum.

I tilfælde af at man skriver kommandoen:

```
fedtmule 11
```

- hvor fedtmule er brugernavn og 11 er produktID, vil der ske følgende:

- ParseCommand vil parse og bestemme at der er tale om et køb.
- Der vil blive tjekket om brugernavn findes, samt om strengen ”11” indeholder et tal, og om det er en gyldig reference til et produkt.
- Er dette overholdt, vil der blive kaldt passende metoder på instansen af Stregsystem, og ud fra resultatet, vil der blive kaldt en passende metode på IStregsystemUI referencen.
- I tilfælde af at købet går godt, vil metoden `DisplayUserBuysProduct` blive kaldt, som vil opdatere brugergrænsefladen med passende tekst.
- Brugergrænsefladen vil nu være klar til at modtage nye kommandoer.

Administration

StregsystemCommandParser skal understøtte administrationskommandoer. Administrationskommandoer er kommandoer der starter med et kolon (':'). Dette er hardcodede kommandoer, der kan bruges til at administrere produkter og lukke ned for systemet. Et eksempel vil være kommandoen `:q (eller :quit)` der skal kunne afslutte programmet (ved kald til Close() på IStregsystemUI instansen)

- StregsystemCommandParser skal definere et dictionary der mapper administratorkommandoer til funktioner. Et andet eksempel på en administratorkommando kunne være ”activate”:
  - `_adminCommands.Add(":activate", (lambda)...Active = true);`
    - `activate`-kommandoen aktiverer et produkt, altså sætter `Active til true`

Følgende administratorkommandoer skal implementeres:

- :quit og :q
  - skal afslutte programmet
- :activate og :deactivate (efterfulgt af produkt-id)
  - skal henholdsvis aktivere og deaktivere et produkt (med mindre det er et seasonal produkt)
- :crediton og :creditoff (efterfulgt af produkt-id)
  - skal henholdsvis aktivere og deaktivere om et produkt kan købes på kredit
- :addcredits (efterfulgt af brugernavn og tal)
  - skal tilføje et antal credits til et brugernavns saldo

Del 4. Implementation og kobling mellem de 3 dele

Det er op til dig at implementere disse kommandoer og hvor i programmet de forskellige dele skal implementeres:

Kommandoer:

- simpelt brugernavn
  - Hvis kommando kun består et brugernavn skal der ske følgende:
    - der skal vises brugernavn, fulde navn og saldo
    - en liste over tidligere køb, op til 10, sorteret, så sidste køb kommer øverst.

        - Hvis saldo er under 50 kr skal brugeren informeres med tekst
        - hvis brugernavnet ikke eksisterer, skal brugeren informeres
- køb af vare
    - Et køb foretages ved at skrive **brugernavn** efterfulgt af **produktnummer**
        - Hvis brugernavnet ikke eksisterer skal brugeren informeres
        - hvis produktet ikke eksisterer skal brugeren informeres
        - hvis produktnummeret ikke er et tal, skal brugeren informeres
        - hvis brugeren ikke har penge, skal et køb ikke gennemføres
- multiköb af varer
    - Multiköb er noget nyt vi vil implementere i stregsystemet:
        - et multiköb foretages ved at skrive **brugernavn** efterfulgt af **antal** (et positivt heltal), og til sidst et gyldigt **produkt-ID**. Dette genererer et antal separate transaktioner, og opdaterer brugergrænsefladen.

En Main-metode, i en Program-klasse, der binder alle 3 dele sammen, kunne se således ud:

```
class Program
{
    static void Main(string[] args)
    {
        IStregsystem stregsystem = new Stregsystem();
        IStregsystemUI ui = new StregsystemCLI(stregsystem);
        StregsystemController sc = new StregsystemController(ui, stregsystem);

        ui.Start();
    }
}
```

Lav et passende klassedesign der opfylder ovenstående krav, inkl. et passende klassehierarki der reducerer koderedundans. Brug ligeledes passende synlighedsmodifikatorer og datatyper. Navngivning af klasser, egenskaber og metoder står jer frit for. Ligeledes står det jer frit for om – og i givet fald hvordan - I vil foretage datavalidering **udover** de beskrevne krav. Endelig står det jer frit for at tilføje yderligere funktionalitet (klasser, metoder og properties) udover de eksplicit angivne krav. I forventes således ikke at understøtte data-persistens.
Opgaven rummer en del variationsmuligheder mht. opfyldelse af de funktionelle krav, hvorfor der ikke kun findes én rigtig måde at løse den på. Der vil dog være fokus på at I laver et hensigtsmæssigt klassedesign, samt at I sørger for at jeres kode er læsbar.
Eventuelle uklarheder i opgaveformuleringen forventes afklaret som en del af opgaveløsningen.