1 Program.cs

```
using System.IO;

namespace Eksamensopgave2017
{
    class Program
    {
        static void Main(string[] args)
        {
            // In the start all files will be loaded and "database" lists will be
             ↪  pupulated
            File.WriteAllText(@"../../transactions.csv", string.Empty);
            ReadFromCsv csvReader = new ReadFromCsv();
            StregsystemCLI ui = new StregsystemCLI();
            csvReader.CreateUserList("../../users.csv");
            csvReader.CreateProductList("../../products.csv");
            csvReader.CreateTransactionList();
            //We will now start the interface
            ui.Start();
        }
    }
}
}
```

2 BuyTransaction.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Eksamensopgave2017
{
    public class BuyTransaction : Transaction
    {
        public BuyTransaction(User user, Product product, DateTime date, decimal
         ↪  amount, List<User> userlist) : base(user, date, amount)
        {
            if (user == null)
                throw new ArgumentNullException();
            else
                this.user = user;

            this.product = product;
```

```
17              this.Date = date;
18              this.Amount = amount;
19              this.UserList = userlist;
20          }
21
22          User user;
23          Product product;
24          DateTime Date;
25          decimal Amount;
26          List<User> UserList;
27
28      public override string ToString()
29      {
30          return $"New purchase  ({this.Id}): {this.user.Firstname}
            ↪  {this.user.Lastname} - {this.Amount} - {this.product.Name} -
            ↪  [{this.Date}]";
31      }
32
33      public override void Execute()
34          {
35              if (this.user.Balance >= this.product.Price ||
                ↪  this.product.CanBeBoughtOnCredit)
36                  {
37                      decimal NewBalance = this.user.Balance - Amount;
38                      foreach (var Item in this.UserList.Where(x => x.Id ==
                        ↪  user.Id))
39                      {
40                          Item.Balance = NewBalance;
41                      }
42                      FileWriters writer = new FileWriters();
43                  //The new information which have been written to the list will
                    ↪  over write the old file
44                      writer.WriteToUserCsv("../../users.csv", UserList);
45                  //The transaction is written to the transaction file
46                      writer.WriteToTransactionCsv("../../transactions.csv",
                        ↪  this.ToString());
47              }
48              else
49              {
50                  return;
51              }
52          }
53      }
54  }
55
56  }
```

3 FileWriters.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4
5  namespace Eksamensopgave2017
6  {
7      class FileWriters
8      {
9          public void WriteToProductCsv(string filepath, List<Product>
           ↪  list)//https://www.codeproject.com/Articles/415732/Reading-and-Writing-CSV-Files-in-Csharp
10         {
11             using (CsvFileWriter writer = new CsvFileWriter(filepath))
12             {
13                 CsvRow FirstRow = new CsvRow();
14                 //First row will be the description row
15                 FirstRow.Add(String.Format("id;name;price;active"));
16                 writer.WriteRow(FirstRow);
17                 CsvRow row = new CsvRow();
18                 //For each row in the file we will now fill the file
19                 foreach (Product element in list)
20                 {
21                     //We need to add it all as strings
22                     string active;
23                     if (element.Active == true)
24                         active = "1";
25                     else
26                         active = "0";
27                     row.Add(element.Id.ToString() + ";\"" +
                       ↪  element.Name.ToString() + "\";" +
                       ↪  element.Price.ToString() + ";" + active + "\n");
28                 }
29                 writer.WriteRow(row);
30             }
31         }
32         public void WriteToUserCsv(string filepath, List<User> list)
33         {
34             using (CsvFileWriter writer = new CsvFileWriter(filepath))
35             {
36                 CsvRow FirstRow = new CsvRow();
37                 //First row will be the description row
38
                   ↪  FirstRow.Add(String.Format("id;firstname;lastname;username;email;balance"));
39                 writer.WriteRow(FirstRow);
40                 CsvRow row = new CsvRow();
41                 //For each row in the file we will now fill the file
```

```
42                  foreach (User element in list)
43                  {
44                      row.Add($"{ element.Id.ToString() };\"{
                        ↪  element.Firstname.ToString() }\";\"{
                        ↪  element.Lastname.ToString() }\";\"{
                        ↪  element.Username.ToString() }\";\"{
                        ↪  element.Email.ToString() }\";{
                        ↪  element.Balance.ToString()} \n");
45                  }
46                  writer.WriteRow(row);
47              }
48          }
49          public void WriteToTransactionCsv(string filepath, string
            ↪  transactioninfo)//https://social.msdn.microsoft.com/Forums/vstudio/en-US/0271c11d-4cf3-452b-af65-6c06
50          {
51              //We create a list, add a single string to it and them writes it to
                ↪  the file
52              List<string> newLines = new List<string>();
53              newLines.Add(transactioninfo.ToString());
54              File.AppendAllLines(filepath, newLines);
55          }
56      }
57  }
58
59  }
```

4 InsertCashTransaction.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace Eksamensopgave2017
8   {
9       public class InsertCashTransaction : Transaction
10      {
11          public InsertCashTransaction(User user, DateTime date, decimal amount,
            ↪  List<User> userlist) : base(user, date, amount)
12          {
13              if (user == null)
14                  throw new ArgumentNullException();
15              else
16                  this.user = user;
```

```
17
18              this.Date = date;
19              this.Amount = amount;
20              this.UserList = userlist;
21          }
22
23          User user;
24          DateTime Date = DateTime.Now;
25          decimal Amount;
26          List<User> UserList;
27
28          public override string ToString()
29          {
30              return $"Cash insertet ({this.Id}): {this.user.Firstname}
                ↪  {this.user.Lastname} - DKK {this.Amount} - [{this.Date}]";
31          }
32
33          public override void Execute()
34          {
35              decimal NewBalance = this.user.Balance + Amount;
36              foreach (var Item in this.UserList.Where(x => x.Id == user.Id))
37              {
38                  Item.Balance = NewBalance;
39              }
40              FileWriters writer = new FileWriters();
41              //The new information which have been written to the list will over
                ↪  write the old file
42              writer.WriteToUserCsv("../../users.csv", UserList);
43              //The transaction is written to the transaction file
44              writer.WriteToTransactionCsv("../../transactions.csv",
                ↪  this.ToString());
45          }
46      }
47  }
48
49  }
```

5 IStregsystem.cs

```
1   using System;
2   using System.Collections.Generic;
3
4   namespace Eksamensopgave2017
5   {
6       public interface IStregsystem
```

```
    {
        IEnumerable<Product> ActiveProducts { get; }
        InsertCashTransaction AddCreditsToAccount(User user, decimal amount,
        ↪  List<User> UserList);
        BuyTransaction BuyProduct(User user, Product product, List<User>
        ↪  UserList, List<Product> ProductList);
        Product GetProductByID(int productID, List<Product> ProductList);
        IEnumerable<Transaction> GetTransactions(User user, int count);
        User GetUser(Func<User, bool> predicate);
        User GetUserByUsername(string username, List<User> UserList);
    }
}
}
```

6 IStregsystemUI.cs

```
namespace Eksamensopgave2017
{
    public interface IStregsystemUI
    {
        void DisplayUserNotFound(string username);
        void DisplayProductNotFound(string product);
        void DisplayUserInfo(User user);
        void DisplayTooManyArgumentsError(string command);
        void DisplayTooFewArgumentsError();
        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 DisplayInactiveProduct(Product product);
        void Start();
    }
}
}
```

7 Product.cs

```
using System;
using System.Collections.Generic;

namespace Eksamensopgave2017
{
    public class Product
    {
        public List<Product> ProductList = new List<Product>();
        public Product(string name, int active, int canbeboughtoncredit, decimal
        ↪  price)
        {
            if (name == null || name == "")
                throw new ArgumentNullException();
            else
                this.Name = name;

            if (active == 1)
                this.Active = true;
            if (active == 0)
                this.Active = false;

            if (canbeboughtoncredit == 1)
                this.CanBeBoughtOnCredit = true;

            this.Price = price;
        }

        public readonly int Id = _nextID++;
        private static int _nextID = 1;
        public string Name;
        public decimal Price;
        public bool Active;
        public bool CanBeBoughtOnCredit = false;

        public override string ToString() //Make other return type
        {
            return $"{this.Name.ToString()} {this.Price.ToString()}
            ↪  ({this.Id.ToString()}) \n";
        }

        public int CompareTo(Product other)
        {
            return this.Name.CompareTo(other.Name);
        }
    }
```

```
}
}
```

8 ReadFromCsv.cs

```
using System;
using System.Collections.Generic;
using System.IO;

namespace Eksamensopgave2017
{
    class ReadFromCsv
    {

        //https://www.codeproject.com/Answers/384264/How-to-read-from-csv-file-using-csharp#answer2
        //https://msdn.microsoft.com/da-dk/library/98f28cdx.aspx
        //Static lists is puplicated so they can be used in all the classes
        //They will work as a "database"
        public static List<Product> ProductList = new List<Product>();
        public static List<SeasonalProduct> SeasonalProductList = new List<SeasonalProduct>();
        public static List<User> UserList = new List<User>();
        public static List<Transaction> TransactionList = new List<Transaction>();
        Stregsystem sSystem = new Stregsystem();
        public void CreateUserList(string fileLocation)
        {
            StreamReader sr = new StreamReader(String.Format(@"{0}", fileLocation));

            string strline = "";
            string[] _values = null;
            int i = 0;
            while (!sr.EndOfStream)
            {
                i++;
                //Remove all " signes
                strline = sr.ReadLine().Replace("\"", "");
                _values = strline.Split(';');
                if (_values.Length == 6 && _values[0].Trim().Length > 0)
                {
                    if (_values[0] == "id")
                    {
                        //Remove the desription line
                    }
```

```
                    else
                    {
                        //Add to list and create an instance of user
                        string firstname = _values[1];
                        string lastname = _values[2];
                        string username = _values[3];
                        string email = _values[4];
                        decimal balance = Convert.ToDecimal(_values[5]);

                        UserList.Add(new User(firstname, lastname, username,
                        ↪  email, balance));
                    }
                }
            }
            sr.Close();
        }

        public void CreateProductList(string fileLocation)
        {
            StreamReader sr = new StreamReader(String.Format(@"{0}",
            ↪  fileLocation));

            string strline = "";
            string[] _values = null;
            int i = 0;
            while (!sr.EndOfStream)
            {
                i++;
                strline = sr.ReadLine();
                //Remove " and HTML tags
                strline = strline.Replace("\"", "").Replace("<h2>",
                ↪  "").Replace("</h2>", "").Replace("<b>", "").Replace("</b>",
                ↪  "").Replace("<h1>", "").Replace("</h1>", "").Replace("<h3>",
                ↪  "").Replace("<h3>", "").Replace("<blink>",
                ↪  "").Replace("</blink>", "");
                _values = strline.Split(';');
                if (_values.Length == 4 && _values[0].Trim().Length > 0)
                {
                    if (_values[0] == "id")
                    {
                        //Remove the desription line
                    }
                    else
                    {
                        string name = _values[1];
                        decimal price = Convert.ToDecimal(_values[2]);
                        int active = 0;
                        if (_values[3] == "1")
```

```
79                          active = 1;
80                      //We need active to be integer, so we change it
81                      ProductList.Add(new Product(name, active, 0, price));
82                  }
83              }
84          }
85          sr.Close();
86          //Pupulate Seasonal ProductList and create instances
87          SeasonalProductList.Add(new SeasonalProduct("Coffe (Black/Latte)", 1,
            ↪  0, 23, 2016, 01, 01, 2017, 09, 01));
88          SeasonalProductList.Add(new SeasonalProduct("Wine (Red)", 1, 0, 456,
            ↪  2016, 01, 01, 2017, 09, 01));
89          SeasonalProductList.Add(new SeasonalProduct("Wine (Wite)", 1, 1, 234,
            ↪  2016, 01, 01, 2017, 01, 01));
90      }
91
92      public void CreateTransactionList()
93      {
94          ////Pupulate transaction list and create instances of both types
95          TransactionList.Add(new
            ↪  BuyTransaction(sSystem.GetUserByUsername("fmikkelsen",
            ↪  ReadFromCsv.UserList), sSystem.GetProductByID(11,
            ↪  ReadFromCsv.ProductList), new DateTime(2012,1,18),
            ↪  sSystem.GetProductByID(11, ReadFromCsv.ProductList).Price,
            ↪  ReadFromCsv.UserList));
96          TransactionList.Add(new
            ↪  InsertCashTransaction(sSystem.GetUserByUsername("bsmith",
            ↪  ReadFromCsv.UserList), new DateTime(2014, 1, 18), 4004,
            ↪  ReadFromCsv.UserList));
97          TransactionList.Add(new
            ↪  BuyTransaction(sSystem.GetUserByUsername("bsmith",
            ↪  ReadFromCsv.UserList), sSystem.GetProductByID(13,
            ↪  ReadFromCsv.ProductList), new DateTime(2016, 6, 18),
            ↪  sSystem.GetProductByID(11, ReadFromCsv.ProductList).Price,
            ↪  ReadFromCsv.UserList));
98          TransactionList.Add(new
            ↪  InsertCashTransaction(sSystem.GetUserByUsername("inielsen",
            ↪  ReadFromCsv.UserList), new DateTime(2016, 8, 18), 4004,
            ↪  ReadFromCsv.UserList));
99          TransactionList.Add(new
            ↪  BuyTransaction(sSystem.GetUserByUsername("inielsen",
            ↪  ReadFromCsv.UserList), sSystem.GetProductByID(15,
            ↪  ReadFromCsv.ProductList), DateTime.Now,
            ↪  sSystem.GetProductByID(11, ReadFromCsv.ProductList).Price,
            ↪  ReadFromCsv.UserList));
100
101         // Execute them to create a transaction file
102         foreach (var element in ReadFromCsv.TransactionList)
```

```
                element.Execute();
            }
        }
    }

}
```

9 SeasonalProduct.cs

```
using System;

namespace Eksamensopgave2017
{
    public class SeasonalProduct : Product
    {
        public DateTime SeasonStartDate;
        public DateTime SeasonEndDate;

        public SeasonalProduct(string name, int active, int canbeboughtoncredit,
         ↪  decimal price, int startYear, int startMonth, int startDay, int
         ↪  endYear, int endMonth, int endDay) : base(name, active,
         ↪  canbeboughtoncredit, price)
        {
            if (name == null || name == "")
                throw new ArgumentNullException();
            else
                this.Name = name;

            this.SeasonStartDate = new DateTime(startYear,startMonth,startDay);
            this.SeasonEndDate = new DateTime(endYear, endMonth, endDay);
            //Check if the product should be active or not
            if (this.SeasonStartDate < DateTime.Now && this.SeasonEndDate >
             ↪  DateTime.Now)
                this.Active = true;
            else
                this.Active = false;

            if (canbeboughtoncredit == 1)
                this.CanBeBoughtOnCredit = true;

            this.Price = price;
        }
    }
}
}
```

10 Stregsystem.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Eksamensopgave2017
{
    public class Stregsystem : IStregsystem
    {
        StregsystemCLI CLI = new StregsystemCLI();

        public IEnumerable<Product> ActiveProducts
        {
            get
            {
                //Combine the to product lists after having checked if the season products should be activated or deactivated
                //It runs after every command, so this will keep them up to date
                List<Product> productList = new List<Product>();
                foreach (SeasonalProduct element in ReadFromCsv.SeasonalProductList)
                {
                    if(element.SeasonStartDate < DateTime.Now && element.SeasonEndDate > DateTime.Now)
                    {
                        element.Active = true;
                    }
                    else
                    {
                        element.Active = false;
                    }
                }

                foreach(var element in ReadFromCsv.SeasonalProductList.FindAll(x => x.Active == true)) productList.Add(element);
                foreach(var element in ReadFromCsv.ProductList.FindAll(x => x.Active == true)) productList.Add(element);
                //http://stackoverflow.com/questions/3386767/linq-orderby-query
                IEnumerable<Product> AktiveProductList = productList.OrderBy(x => x.ToString()).ToList();
                return AktiveProductList;
            }
        }

        public InsertCashTransaction AddCreditsToAccount(User user, decimal amount, List<User> UserList)
```

```
39          {
40              InsertCashTransaction Transaction = new InsertCashTransaction(user,
                ↪  DateTime.Now, amount, UserList);
41              return Transaction;
42          }
43
44          public BuyTransaction BuyProduct(User user, Product product, List<User>
            ↪  UserList, List<Product> ProductList)
45          {
46              BuyTransaction Transaction = new BuyTransaction(user, product,
                ↪  DateTime.Now, product.Price, UserList);
47              return Transaction;
48          }
49
50          public Product GetProductByID(int productID, List<Product> ProductList)
51          {
52              //Check if the product exists. If it does check if it is active
53              if(ProductList.Exists(x => x.Id == productID))
54              {
55                  Product product = ProductList.Find(x => x.Id == productID);
56                  if (product.Active)
57                  {
58                      return product;
59                  }
60                  else
61                  {
62                      throw new InactiveProductException();
63                  }
64
65              }
66              else
67              {
68                  throw new ProductDoesNotExsistException();
69              }
70
71          }
72
73          public IEnumerable<Transaction> GetTransactions(User user, int count)
74          {
75              IEnumerable<Transaction> transactions =
                ↪  ReadFromCsv.TransactionList.Where(x => x.user ==
                ↪  user).OrderByDescending(x => x.Date).Take(count);
76              return transactions;
77          }
78
79          public User GetUserByUsername(string username, List<User> UserList)
80          {
81
```

```
82              if (UserList.Exists(x => x.Username == username))
83              {
84                  User user = UserList.Find(x => x.Username == username);
85                  return user;
86              }
87              else
88              {
89                  throw new InvalidUsernameException();
90              }
91          }
92
93          public User GetUser(Func<User, bool> predicate)
94          {
95              throw new NotImplementedException();
96          } //Not Done Yet
97      }
98  }
99
100 }
```

11 StregsystemCLI.cs

```
1   using System;
2
3   namespace Eksamensopgave2017
4   {
5
6       public class StregsystemCLI : IStregsystemUI
7       {
8           public void Close()
9           {
10              _running = false;
11          }
12
13          public void DisplayAdminCommandNotFoundMessage(string adminCommand)
14          {
15              Console.WriteLine($"The admin command '{adminCommand}' could not be
                ↪  found");
16          }
17
18          public void DisplayGeneralError(string errorString)
19          {
20              throw new NotImplementedException();
21          }
22
```

```
23          public void DisplayInactiveProduct(Product product)
24          {
25              Console.WriteLine($"'{product.Name}' is inactive (ID:
                ↪  {product.Id})");
26          }
27
28          public void DisplayInsufficientCash(User user, Product product)
29          {
30              Console.WriteLine($"{user.Firstname} {user.Lastname} does not have
                ↪  sufficient cash to buy {product.Name} \n{user.Firstname} have a
                ↪  balance of {user.Balance}");
31          }
32
33          public void DisplayProductNotFound(string product)
34          {
35              Console.WriteLine($"Product not found");
36          }
37
38          public void DisplayTooManyArgumentsError(string command)
39          {
40              Console.WriteLine($"The following command contans to many arguments:
                ↪  {command}");
41          }
42
43          public void DisplayTooFewArgumentsError()
44          {
45              Console.WriteLine($"To few arguments");
46          }
47
48          public void DisplayUserBuysProduct(BuyTransaction transaction)
49          {
50              Console.WriteLine($"{transaction.user.Firstname} bought a
                ↪  product\nUser balance: {transaction.user.Balance}");
51          }
52
53          public void DisplayUserInfo(User user)
54          {
55              Stregsystem sSystem = new Stregsystem();
56              //http://stackoverflow.com/a/5344836
57              //http://stackoverflow.com/a/10883477
58              Console.WriteLine($"{user.Username}, {user.Firstname}
                ↪  {user.Firstname}, balance: {user.Balance}");
59              foreach (var element in sSystem.GetTransactions(user, 10))
60              {
61                  Console.WriteLine(element);
62              }
63          }
```

```
64
65          public void DisplayUserNotFound(string username)
66          {
67              Console.WriteLine($"User {username} does not exist");
68          }
69
70          public void DisplayAktiveProducts()
71          {
72              Console.Clear();
73              Stregsystem ssystem = new Stregsystem();
74              foreach (var Item in ssystem.ActiveProducts)
75              {
76                  Console.Write(Item.ToString());
77              }
78              Console.WriteLine();
79          }
80          private bool _running;
81          public void Start()
82          {
83
84              StregsystemController ssc = new StregsystemController();
85              _running = true;
86              DisplayAktiveProducts();
87              Console.Write("\nQuickbuy: ");
88              do
89              {
90                  string command = Console.ReadLine();
91                  DisplayAktiveProducts();
92                  if (command == ":q" || command == ":quit")
93                      Close();
94                  ssc.RunCommand(command);
95                  Console.Write("\n\nQuickbuy: ");
96
97              } while (_running);
98          }
99
100         public void DisplayUserBuysProduct(int count, BuyTransaction transaction)
101         {
102             Console.WriteLine($"{transaction.user.Firstname} bought {count}
                ↪  products \nUser balance: {transaction.user.Balance}");
103         }
104
105         public void LowBalanceWarning(User user, decimal amount)
106         {
107             if (user.Balance < amount) Console.WriteLine("!!---> YOUR BALANCE IS
                ↪  UNDER 50 <---!!");
108         }
109     }
```

```
}
}
```

12 StregsystemController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;

namespace Eksamensopgave2017
{
    class StregsystemController
    {

        //http://stackoverflow.com/questions/2829873/how-can-i-detect-if-this-dictionary-key-exists-in-c
        //https://www.tutorialspoint.com/csharp/csharp_delegates.htm
        //https://msdn.microsoft.com/en-us/library/bb882516.aspx

        //http://stackoverflow.com/questions/4233536/c-sharp-store-functions-in-a-dictionary

        //http://stackoverflow.com/questions/2896715/dictionary-with-delegate-or-switch
        //http://stackoverflow.com/a/21099511

        delegate List<Product> StatusCommands(int productId);
        delegate void QuitCommand();
        delegate List<Product> CreditCommands(int productId);
        delegate InsertCashTransaction AddCreditsCommands(User user, decimal amount, List<User> UserList);

        StregsystemCLI CLI = new StregsystemCLI();
        Stregsystem sSystem = new Stregsystem();

        public Dictionary<string, Delegate> _adminCommands = new Dictionary<string, Delegate>();

        public static List<Product> Aktivate(int productId)
        {
            StregsystemCLI CLI = new StregsystemCLI();
            FileWriters writer = new FileWriters();
            ReadFromCsv.ProductList.Where(x => x.Id == productId).ToList().ForEach(x => x.Active = true);
            writer.WriteToProductCsv("../../products.csv", ReadFromCsv.ProductList);
            CLI.DisplayAktiveProducts();
```

```
34              return null;
35          }
36
37          public static List<Product> Deactivate(int productId)
38          {
39              StregsystemCLI CLI = new StregsystemCLI();
40              FileWriters writer = new FileWriters();
41              ReadFromCsv.ProductList.Where(x => x.Id ==
                ↪  productId).ToList().ForEach(x => x.Active = false);
42              writer.WriteToProductCsv("../../products.csv",
                ↪  ReadFromCsv.ProductList);
43              CLI.DisplayAktiveProducts();
44              return null;
45          }
46
47          public static List<Product> CreaditOn(int productId)
48          {
49              FileWriters writer = new FileWriters();
50              ReadFromCsv.ProductList.Where(x => x.Id ==
                ↪  productId).ToList().ForEach(x => x.CanBeBoughtOnCredit = true);
51              writer.WriteToProductCsv("../../products.csv",
                ↪  ReadFromCsv.ProductList);
52              return null;
53          }
54
55          public static List<Product> CreditOff(int productId)
56          {
57              FileWriters writer = new FileWriters();
58              ReadFromCsv.ProductList.Where(x => x.Id ==
                ↪  productId).ToList().ForEach(x => x.CanBeBoughtOnCredit = false);
59              writer.WriteToProductCsv("../../products.csv",
                ↪  ReadFromCsv.ProductList);
60              return null;
61          }
62
63          public StregsystemController()
64          {
65              StatusCommands MakeAktive = new StatusCommands(Aktivate);
66              StatusCommands MakeDeactivacted = new StatusCommands(Deactivate);
67              QuitCommand Quit = new QuitCommand(CLI.Close);
68              CreditCommands MakeCreditOn = new CreditCommands(CreaditOn);
69              CreditCommands MakeCreditOff = new CreditCommands(CreditOff);
70              AddCreditsCommands AddCredits = new
                ↪  AddCreditsCommands(sSystem.AddCreditsToAccount);
71
72              _adminCommands.Add(":activate", MakeAktive);
73              _adminCommands.Add(":deactivate", MakeDeactivacted);
74              _adminCommands.Add(":quit", Quit);
```

```
75              _adminCommands.Add(":q", Quit);
76              _adminCommands.Add(":crediton", MakeCreditOn);
77              _adminCommands.Add(":creditoff", MakeCreditOff);
78              _adminCommands.Add(":addcredits", AddCredits);
79          }
80
81          public void RunCommand(string command)
82          {
83              if (command == "")
84                  CLI.DisplayTooFewArgumentsError();
85              else if (Convert.ToString(command[0]) == ":") //If it starts with :
                ↪  we know it is a admin command
86              {
87                  string[] SplittedString = command.Split(' '); //Split it so we
                    ↪  can see which command and use the info
88                  string AdminCommand = Convert.ToString(SplittedString[0]);
89                  if (_adminCommands.ContainsKey(AdminCommand));
90                  {
91                      if (AdminCommand == ":q" || AdminCommand == ":quit")
92                      {
93                          _adminCommands[":q"].DynamicInvoke();
94                      }
95                      else if (AdminCommand == ":activate" || AdminCommand ==
                        ↪  ":deactivate" || AdminCommand == ":crediton" ||
                        ↪  AdminCommand == ":creditoff")
96                      {
97                          int productId = Convert.ToInt32(SplittedString[1]);
98                          _adminCommands[AdminCommand].DynamicInvoke(productId);
99                      }
100                     else if (AdminCommand == ":addcredits")
101                     {
102                         string username = Convert.ToString(SplittedString[1]);
103                         User user = sSystem.GetUserByUsername(username,
                            ↪  ReadFromCsv.UserList);
104                         decimal amount = Convert.ToDecimal(SplittedString[2]);
105                         sSystem.AddCreditsToAccount(user, amount,
                            ↪  ReadFromCsv.UserList).Execute();
106                     }
107                     else
108                     {
109                         CLI.DisplayAdminCommandNotFoundMessage(command);
110                     }
111                 }
112             }
113             else
114             {
115                 //Setup Regex to notice the tree different kindes if commands
```

```
116                 Match username = Regex.Match(command, @"^[a-z\d_-]+$"); //Only
                    ↪  username
117                 Match buy = Regex.Match(command, @"^[a-z\d_-]+ \d+$"); //username
                    ↪  numbers
118                 Match multibuy = Regex.Match(command, @"^[a-z\d_-]+ \d+ \d+$");
                    ↪  //username numbers numbers
119
120                 if (username.Success)
121                 {
122                     try
123                     {
124                         User user = sSystem.GetUserByUsername(command,
                            ↪  ReadFromCsv.UserList);
125                         CLI.DisplayUserInfo(user);
126                     }
127                     catch (InvalidUsernameException)
128                     {
129                         CLI.DisplayUserNotFound(command);
130                     }
131                 }
132                 else if (buy.Success)
133                 {
134                     string[] buyArray = command.Split(' '); //Split the command
                        ↪  to  get the information
135                     try
136                     {
137                         User user =
                            ↪  sSystem.GetUserByUsername(Convert.ToString(buyArray[0]),
                            ↪  ReadFromCsv.UserList);
138                         Product product =
                            ↪  sSystem.GetProductByID(Convert.ToInt32(buyArray[1]),
                            ↪  ReadFromCsv.ProductList);
139                         if (user.Balance >= product.Price ||
                            ↪  product.CanBeBoughtOnCredit)
140                         {
141                             sSystem.BuyProduct(user, product,
                                ↪  ReadFromCsv.UserList,
                                ↪  ReadFromCsv.ProductList).Execute();
142                             CLI.DisplayUserBuysProduct(sSystem.BuyProduct(user,
                                ↪  product, ReadFromCsv.UserList,
                                ↪  ReadFromCsv.ProductList));
143                             CLI.LowBalanceWarning(user, 50); // Check for low
                                ↪  balance
144                         }
145                         else
146                         {
147                             CLI.DisplayInsufficientCash(user, product);
148                         }
```

```
149                     }
150                     catch (InvalidUsernameException) {
                        ↪  CLI.DisplayUserNotFound(Convert.ToString(buyArray[0]));
                        ↪  }
151                     catch (InactiveProductException)
152                     {
153                         //Create a list of all inactive products to finde the
                            ↪  specifc product by id
154                         List<Product> productList = new List<Product>();
155                         foreach (var element in
                            ↪  ReadFromCsv.SeasonalProductList.FindAll(x =>
                            ↪  x.Active == false)) productList.Add(element);
156                         foreach (var element in ReadFromCsv.ProductList.FindAll(x
                            ↪  => x.Active == false)) productList.Add(element);
157                         Product product = productList.Find(x => x.Id ==
                            ↪  Convert.ToInt32(buyArray[1]));
158                         CLI.DisplayInactiveProduct(product);
159                     }
160                     catch (ProductDoesNotExsistException) {
                        ↪  CLI.DisplayProductNotFound(null); }
161                 }
162                 else if (multibuy.Success)
163                 {
164                     string[] buyArray = command.Split(' ');
165                     try
166                     {
167                         User user =
                            ↪  sSystem.GetUserByUsername(Convert.ToString(buyArray[0]),
                            ↪  ReadFromCsv.UserList);
168                         int productCount = Convert.ToInt32(buyArray[1]);
169                         Product product =
                            ↪  sSystem.GetProductByID(Convert.ToInt32(buyArray[2]),
                            ↪  ReadFromCsv.ProductList);
170                         decimal totalPrice = productCount * product.Price;
171                         if ((user.Balance >= totalPrice ||
                            ↪  product.CanBeBoughtOnCredit))
172                         {
173                             for (int i = 0; i < productCount; i++) // Run the
                                ↪  command once for each bought product
174                                 sSystem.BuyProduct(user, product,
                                    ↪  ReadFromCsv.UserList,
                                    ↪  ReadFromCsv.ProductList).Execute();
175                             CLI.DisplayUserBuysProduct(productCount,
                                ↪  sSystem.BuyProduct(user, product,
                                ↪  ReadFromCsv.UserList, ReadFromCsv.ProductList));
176                             CLI.LowBalanceWarning(user, 50);
177                         }
178                         else
```

```
179                         {
180                             CLI.DisplayInsufficientCash(user, product);
181                         }
182                     }
183 
                        ↪  //http://stackoverflow.com/questions/136035/catch-multiple-exceptions-at-once
184                     catch (InvalidUsernameException) {
                        ↪  CLI.DisplayUserNotFound(Convert.ToString(buyArray[0]));
                        ↪  }
185                     catch (InactiveProductException)
186                     {
187                         List<Product> productList = new List<Product>();
188                         foreach (var element in
                            ↪  ReadFromCsv.SeasonalProductList.FindAll(x =>
                            ↪  x.Active == false)) productList.Add(element);
189                         foreach (var element in ReadFromCsv.ProductList.FindAll(x
                            ↪  => x.Active == false)) productList.Add(element);
190                         Product product = productList.Find(x => x.Id ==
                            ↪  Convert.ToInt32(buyArray[2]));
191                         CLI.DisplayInactiveProduct(product);
192                     }
193                     catch (ProductDoesNotExsistException) {
                        ↪  CLI.DisplayProductNotFound(null); }
194 
195                 }
196                 else
197                 {
198                      CLI.DisplayTooManyArgumentsError(command);
199                 }
200             }
201         }
202     }
203 }
204 
205 }
```

13 Transaction.cs

```
using System;

namespace Eksamensopgave2017
{
    public class Transaction
    {
        public Transaction(User user, DateTime date, decimal amount)
        {
            if (user == null)
                throw new ArgumentNullException();
            else
                this.user = user;
            this.Date = date;
            this.Amount = amount;
        }

        public readonly int Id = _nextID++;
        private static int _nextID = 1;
        public User user;
        public DateTime Date;
        public decimal Amount;

        public override string ToString()
        {
            return $"New transaction ({this.Id}): {this.user.Firstname}
             ↪  {this.user.Lastname} - DKK {this.Amount} - [{this.Date}]";
        }

        public virtual void Execute()
        {
        }
    }
}

}
```

14 User.cs

```
using System;
using System.Text.RegularExpressions;

namespace Eksamensopgave2017
{
    public class User : IComparable<User>
    {
        //https://msdn.microsoft.com/en-us/library/twcw2f1c(v=vs.110).aspx
        Regex UsernameValidator = new Regex(@"^[a-z\d_-]+$");
        Regex EmailLocalValidator = new Regex(@"^[a-zA-Z\d_\-.]+$");
        Regex EmailDomainValidator = new Regex(@"^[a-zA-Z\d\-.]+$");
        public User(string firstname, string lastname, string username, string
        ↪  email, decimal balance)
        {
            string ConvertedUsername = username.Replace("\"", "");
            ↪  //http://stackoverflow.com/a/1177897
            Match mUsername = UsernameValidator.Match(ConvertedUsername);
            if (mUsername.Success)
                this.Username = ConvertedUsername;
            else
                throw new InvalidUsernameException();

            string[] SplittedEmail = email.Split('@'); // Split it to find local
            ↪  and domain
            Match mLocal = EmailLocalValidator.Match(SplittedEmail[0]);
            Match mDomain = EmailDomainValidator.Match(SplittedEmail[1]);
            int DomainLength = SplittedEmail[1].Length - 1;
            //If both the Regex validators is okay and the char check does not
            ↪  match we asign email
            if (mLocal.Success && mDomain.Success && SplittedEmail[1][0] !=
            ↪  Convert.ToChar(".") && SplittedEmail[1][0] !=
            ↪  Convert.ToChar("-") && SplittedEmail[1][DomainLength] !=
            ↪  Convert.ToChar(".") && SplittedEmail[1][DomainLength] !=
            ↪  Convert.ToChar("-"))
                this.Email = email;
            else
                throw new InvalidEmailException();

            this.Balance = balance;

            if (firstname == null || firstname == "" || lastname == null ||
            ↪  lastname == "")
                throw new ArgumentNullException();
            else
            {
```

```
37                  this.Firstname = firstname;
38                  this.Lastname = lastname;
39              }
40
41          }
42
43          public readonly int Id = _nextID++;
44          private static int _nextID = 1;
45          public string Firstname;
46          public string Lastname;
47          public string Username;
48          public string Email;
49          public decimal Balance;
50
51
            ↪  //https://msdn.microsoft.com/en-us/library/system.object.tostring(v=vs.110).aspx
52          public override string ToString()
53          {
54              return $"{this.Firstname.ToString()} {this.Lastname.ToString()}\n";
55          }
56
57          public override bool Equals(object obj)
58          {
59              User other = obj as User;
60              if (other == null)
61                  return false;
62              return Id.Equals(other.Id);
63          }
64
65          public override int GetHashCode()
66          {
67              return base.GetHashCode();
68          }
69
70          public int CompareTo(User other)
71          {
72              return this.Id.CompareTo(other.Id);
73          }
74      }
75  }
76
77  }
```

15 WriteToCsv.cs

```
using System.Collections.Generic;
using System.IO;
using System.Text;

namespace Eksamensopgave2017 // https://www.codeproject.com/Articles/415732/Reading-and-Writing-CSV-Files-in-Csharp
{
    public class CsvRow : List<string>
    {
        public string LineText { get; set; }
    }

    public class CsvFileWriter : StreamWriter
    {
        public CsvFileWriter(Stream stream) : base(stream)
        {
        }

        public CsvFileWriter(string filename) : base(filename)
        {
        }

        public void WriteRow(CsvRow row)
        {
            StringBuilder builder = new StringBuilder();
            foreach (string value in row)
            {
                builder.Append(value);
            }
            row.LineText = builder.ToString();
            WriteLine(row.LineText);
        }
    }
}

}
```