# Model

## Class Date

using System;

namespace CourseWork.Model

{

//Дата {Місяць, День};

[Serializable()]

class Date: ISpreadsheet

{

public const int MinDay = 1;

public const int MinMonth = 1;

public const int MaxMonth = 12;

private int day;

private int month = MinMonth; // 1 - 12, Jan - Dec

public Date(string day, string month)

: this(Int32.Parse(day), Int32.Parse(month))

{

}

public Date(int day, int month)

{

Day = day;

Month = month;

}

public Date(Date date)

{

this.day = date.day;

this.month = date.month;

}

public int Day

{

get

{

return day;

}

protected set

{

if (value <= MaxDayMonth(month) && value >= MinDay)

{ day = value; }

else

{

throw new ArgumentOutOfRangeException(SupportText.DateDay,

SupportText.ArgumentOutOfRange + ": " + MinDay + '-'

+ MaxDayMonth(month).ToString());

}

}

}

public int Month

{

get

{

return month;

}

protected set

{

if (value >= MinMonth && value <= MaxMonth)

{ month = value; }

else

{

throw new ArgumentOutOfRangeException(SupportText.DateMonth,

SupportText.ArgumentOutOfRange + ": " + MinMonth + '-'

+ MaxMonth);

}

}

}

public static int MaxDayMonth(int month)

{

if (month > MaxMonth || month < MinMonth)

{

throw new ArgumentOutOfRangeException(

SupportText.DateMonth,

SupportText.ArgumentOutOfRange);

}

int max;

switch (month)

{

case 4:

case 6:

case 9:

case 11: max = 30; break;

case 2: max = 28; break;

default: max = 31; break;

}

return max;

}

public static bool operator <(Date obj1, Date obj2)

{

if (obj1 == null)

return true;

else if (obj2 == null)

return false;

return (obj1.month < obj2.month || (obj1.month == obj2.month && obj1.day < obj2.day));

}

public static bool operator >(Date obj1, Date obj2)

{

if (obj1 == null)

return false;

else if (obj2 == null)

return true;

return (obj1.month > obj2.month || (obj1.month == obj2.month && obj1.day > obj2.day));

}

public string ToTableRow()

{

const string Format = "{0, 2:00}.{1, 2:00}\u2502";

return string.Format(Format, day, month);

}

public override string ToString()

{

return day + ' '.ToString() + month;

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

Date date = obj as Date;

if (date == null)

{ return false; }

return (date.day == day && Month == date.month);

}

}

}

## Class Seller

using System;

namespace CourseWork.Model

{

//Продавець {Прізвище, Ім’я, Дата (прийняття на роботу)};

[Serializable()]

class Seller: ISpreadsheet

{

private string name;

private string surname;

private Date recruited;

public Seller(string name, string surname, Date date)

{

Name = name;

Surname = surname;

recruited = new Date(date);

}

public Seller(string day, string month, string name, string surname)

{

Name = name;

Surname = surname;

recruited = new Date(day, month);

}

public string Name

{

get

{

return name;

}

set

{

if (String.IsNullOrEmpty(value))

{

throw new ArgumentNullException(SupportText.SellerName,

SupportText.ArgumentNull);

}

else

{

name = value;

}

}

}

public string Surname

{

get

{

return surname;

}

set

{

if (String.IsNullOrEmpty(value))

{

throw new ArgumentNullException(SupportText.SellerSurname,

SupportText.ArgumentNull);

}

else

{ surname = value; }

}

}

internal Date Recruited

{

get

{

return recruited;

}

private set

{

if (value == null)

throw new ArgumentNullException(SupportText.Date,

SupportText.ArgumentNull);

recruited = value;

}

}

public override string ToString()

{

return recruited.ToString() + ' ' + Name + ' ' + Surname;

}

public new bool Equals(object obj)

{

if (obj == null)

return false;

Seller seller = obj as Seller;

if (seller == null)

{ return false; }

return (Name.Equals(seller.Name) && Surname.Equals(seller.Surname)

&& recruited.Equals(seller.recruited));

}

public string ToTableRow()

{

const string Format = "{0, 14}\u2502";

return string.Format(Format, Surname);

}

}

}

## Class Goods

using System;

namespace CourseWork.Model

{

//Товар {Назва, Дата (надходження), Тип (Ваговий, розпакований), Ціна}

/// <summary>

/// Перелічування стану товарів

/// </summary>

public enum GoodsType { Unpacked, WeighedOut }

[Serializable()]

class Goods : ISpreadsheet

{

private string name; // Ім'я товару

private double price; // Ціна товару

private Date receiving;

public Goods(string name, double price, Date date, GoodsType type)

{

Name = name;

Price = price;

Receiving = date;

Type = type;

}

public Goods(string day, string month, string name, string price, string type)

{

Name = name;

Price = Double.Parse(price);

Type = (GoodsType)Enum.Parse(typeof(GoodsType), type);

Receiving = new Date(day, month);

}

public Goods(Goods goods)

{

this.name = goods.name;

this.price = goods.price;

receiving = new Date(goods.Receiving);

}

public GoodsType Type

{

get; protected set;

}

public double Price

{

get

{ return price; }

protected set

{

if (value > 0.0)

{ price = value; }

else

{ throw new ArgumentOutOfRangeException(

SupportText.Price, SupportText.ArgumentOutOfRange

+ ": 0.0 <"); }

}

}

public string Name

{

get

{

return name;

}

protected set

{

if (String.IsNullOrEmpty(value)) // якщо null або ""

{

throw new ArgumentNullException(SupportText.GoodsName,

SupportText.ArgumentNull);

}

else

{ name = value; }

}

}

public Date Receiving

{

get

{

return receiving;

}

private set

{

if (value == null)

throw new ArgumentNullException(SupportText.Date,

SupportText.ArgumentNull);

receiving = value;

}

}

public override string ToString()

{

return receiving.ToString() + ' ' + name + ' ' + price + ' ' + Enum.GetName(typeof(GoodsType), Type);

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

Goods goods = obj as Goods;

if (goods == null)

return false;

return (receiving.Equals(goods.receiving) && price == goods.price

&& Name.Equals(goods.Name) && Type == goods.Type);

}

/// <summary>

public string ToTableRow()

{

const string Format = "{0, 14}\u2502{1, 11}\u2502{2,7:##0.0}\u2502";

return String.Format(Format, name, (Type == GoodsType.Unpacked)

? SupportText.GoodsTypeUnpacked

: SupportText.GoodsTypeWeighedOut, price);

}

}

}

## Class Sale

using System;

namespace CourseWork.Model

{

//Операція Продажу { Дата (продажу), Продавець, Товар, Кількість товару };

[Serializable()]

class Sale : ISpreadsheet

{

private double amount = 1; // кількість товару

private Date saleDate;

private Goods goodsTransaction;

private Seller seller;

public Sale(Seller seller, Goods goods, Date saleDate, double amount)

{

SellerTransaction = seller;

GoodsTransaction = goods;

SaleDate = saleDate;

Amount = amount;

}

public Sale(string saleDay, string saleMonth, string sellerEmployDay,

string sellerEmployMonth, string sellerName, string sellerSurname,

string goodsReceiptDay, string goodsReceiptMonth,

string goodsName, string goodsPrice, string goodsType,

string amount)

{

SellerTransaction = new Seller(sellerEmployDay, sellerEmployMonth,

sellerName, sellerSurname);

goodsTransaction = new Goods(goodsReceiptDay, goodsReceiptMonth,

goodsName, goodsPrice, goodsType);

SaleDate = new Date(saleDay, saleMonth);

Amount = Double.Parse(amount);

}

internal Seller SellerTransaction

{

get

{

return seller;

}

private set

{

if (value == null)

throw new ArgumentNullException(SupportText.Goods,

SupportText.ArgumentNull);

seller = value;

}

}

internal Goods GoodsTransaction

{

get

{

return goodsTransaction;

}

private set

{

if (value == null)

throw new ArgumentNullException(SupportText.Goods,

SupportText.ArgumentNull);

goodsTransaction = new Goods(value);

}

}

internal Date SaleDate

{

get

{

return saleDate;

}

private set

{

if (value == null)

throw new ArgumentNullException(SupportText.Date,

SupportText.ArgumentNull);

saleDate = value;

}

}

public double Amount

{

get

{

return amount;

}

set

{

if (value > 0.0)

{ amount = value; }

else

{

throw new ArgumentOutOfRangeException(

SupportText.SaleAmount, SupportText.ArgumentOutOfRange

+ ": 0.0 <");

}

}

}

public double TotalPrice

{

get

{

return amount \* GoodsTransaction.Price;

}

}

public override string ToString()

{

return SaleDate.ToString() + ' ' + SellerTransaction.ToString() + ' ' + GoodsTransaction.ToString() + ' ' + Amount;

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

Sale sale = obj as Sale;

if (sale == null)

return false;

if (sale == this)

{

return true;

}

return (amount == sale.amount && saleDate.Equals(sale.saleDate)

&& SellerTransaction.Equals(sale.SellerTransaction)

&& GoodsTransaction.Equals(sale.GoodsTransaction));

}

public string ToTableRow()

{

return '\u2502' + saleDate.ToTableRow() + GoodsTransaction.ToTableRow() + String.Format("{0, 7: 0.0}\u2502{1, 14:0.0}\u2502", amount, TotalPrice) + SellerTransaction.ToTableRow();

}

}

}

## Class Balance

using System;

using System.Collections.Generic;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

namespace CourseWork.Model

{

[Serializable()]

class Balance : IObservable, ISpreadsheet

{

List<Sale> balance; //список операцій

public delegate bool Compare(Sale sale, object obj);

public delegate Sale Statistics();

protected event NotifyObserver send;

public Balance()

{

balance = new List<Sale>();

}

public event NotifyObserver Send

{

add

{

if (value != null)

{

send += value;

}

}

remove

{

if (value != null)

{

send -= value;

}

}

}

public int Length

{

get

{

return balance.Count;

}

}

public double TotalCash

{

get

{

double totalCash = 0.0;

foreach (Sale sale in balance)

{

totalCash += sale.TotalPrice;

}

return totalCash;

}

}

public void WriteToBinaryFile(string fileName)

{

Stream stream = null;

BinaryFormatter writer;

try

{

stream = File.Open(fileName, FileMode.OpenOrCreate, FileAccess.Write);

writer = new BinaryFormatter();

writer.Serialize(stream, balance);

}

finally

{

if (stream != null)

{ stream.Close(); }

}

}

public static List<Sale> ReadFromBinaryFile(string fileName)

{

List<Sale> readInformation;

Stream stream = null;

try

{

stream = File.Open(fileName, FileMode.Open);

BinaryFormatter bFormatter = new BinaryFormatter();

readInformation = (List<Sale>)bFormatter.Deserialize(stream);

}

finally

{

if (stream != null)

{

stream.Close();

}

}

return readInformation;

}

public static List<Sale> ReadFromTextFile(string fileName)

{

StreamReader reader = null;

List<Sale> readInformation = null;

Sale record;

string [] inputFields;

try

{

reader = File.OpenText(fileName);

readInformation = new List<Sale>();

while (reader.Peek() != -1) // поки не кінець файлу

{

inputFields = reader.ReadLine().Split(' ');

record = new Sale(inputFields[0], inputFields[1],

inputFields[2], inputFields[3], inputFields[4],

inputFields[5], inputFields[6], inputFields[7],

inputFields[8], inputFields[9], inputFields[10],

inputFields[11]);

readInformation.Add(record);

}

}

finally

{

if (reader != null)

{

reader.Close();

}

}

return readInformation;

}

public void WriteToTextFile(string fileName)

{

StreamWriter writer = null;

try

{

writer = new StreamWriter(fileName);

foreach (Sale element in balance)

{

writer.WriteLine(element.ToString());

}

}

finally

{

if (writer != null)

{ writer.Close(); }

}

}

public Sale this[int index]

{

get

{

if (index >= 0 && index < Length)

{ return balance[index]; }

else

{ throw new IndexOutOfRangeException("Індекс колекції за межами діапазону"); }

}

}

public void Add(Sale sale)

{

if (sale != null)

{

if (balance.Contains(sale))

return;

int i = 0;

while (i < balance.Count && balance[i].SaleDate < sale.SaleDate)

{

i++;

}

balance.Insert(i, sale);

Notify();

}

else

{

throw new ArgumentNullException();

}

}

public void Notify()

{

if (send != null)

send();

}

public Balance Search(Compare del, object obj)

{

Balance sales = new Balance();

foreach (Sale element in balance)

{

if (del(element, obj))

{

sales.Add(element);

}

}

return sales;

}

public bool CompareByName(Sale sale, object name)

{

string field = (name as string).ToUpper();

return (sale.GoodsTransaction.Name.ToUpper().Equals(field));

}

public bool CompareByPrice(Sale sale, object price)

{

return (sale.GoodsTransaction.Price.Equals(price));

}

#endregion

public Sale Max()

{

Sale max;

try

{

max = balance[0];

}

catch

{

throw;

}

foreach (Sale sale in balance)

{

if (sale.TotalPrice > max.TotalPrice)

max = sale;

}

return max;

}

public Sale Min()

{

Sale min;

try

{

min = balance[0];

}

catch

{

throw;

}

foreach (Sale sale in balance)

{

if (sale.TotalPrice < min.TotalPrice)

min = sale;

}

return min;

}

public string ToTableRow()

{

StringBuilder rows = new StringBuilder();

foreach (Sale element in balance)

{

rows.Append(element.ToTableRow());

}

return rows.ToString();

}

}

}

## Interface ITable

namespace CourseWork.Model

{

interface ISpreadsheet

{

string ToTableRow();

}

}

# View

## Abstract class Menu

using System;

namespace CourseWork.View

{

>

abstract class Menu

{

private int previousPos;

private int currentPos;

private int min;

private int max;

protected string[] menu;

public const int ExitCode = -1;

public Menu(ConsoleColor font, ConsoleColor background, params string [] menu)

:this(font, background)

{

if (menu != null)

{

this.menu = new string[menu.Length + 1];

for (int i = 0; i < menu.Length; i++)

{

if (String.IsNullOrEmpty(menu[i]))

{ throw new ArgumentOutOfRangeException(); }

else

{ this.menu[i] = menu[i]; }

}

}

else

{

throw new ArgumentNullException();

}

}

protected Menu(ConsoleColor font, ConsoleColor background)

{

Font = font;

Background = background;

}

public ConsoleColor Font

{

get; protected set;

}

public ConsoleColor Background

{

get; protected set;

}

private int Position

{

get

{

return currentPos;

}

set

{

previousPos = currentPos;

if (value < min)

currentPos = max;

else if (value > max)

currentPos = min;

else

currentPos = value;

}

}

public int Choice

{

get

{

if (currentPos - min == menu.Length - 1)

return ExitCode;

return currentPos - min;

}

}

private void PrintMenu()

{

for (int i = 0; i < menu.Length; i++)

{

Console.WriteLine(menu[i]);

}

}

private void ChangeColors(ConsoleColor background, ConsoleColor foreground)

{

Console.BackgroundColor = background;

Console.ForegroundColor = foreground;

}

private void SetActiveMenuItem()

{

Console.CursorVisible = false;

ChangeRowColors(currentPos, Background, Font);

ConsoleKeyInfo pressedKey;

do

{

// previousPos = currentPos;

pressedKey = Console.ReadKey(true);

switch (pressedKey.Key)

{

case ConsoleKey.UpArrow:

{ Position--; }

break;

case ConsoleKey.DownArrow:

{ Position++; }

break;

}

ChangeRowColors(previousPos, Font, Background);

ChangeRowColors(currentPos, Background, Font);

} while (pressedKey.Key != ConsoleKey.Enter);

Console.CursorVisible = true;

}

private void ChangeRowColors(int cursorTop, ConsoleColor foreground, ConsoleColor background)

{

if (cursorTop >= min && cursorTop <= max)

{

ChangeColors(background, foreground);

ViewComponent.ClearString(cursorTop);

Console.WriteLine(menu[cursorTop - min]);

}

}

protected int DoMenuAction(string msg)

{

ViewComponent.ClearWithotFirst();

ChangeColors(Background, Font);

Console.WriteLine(StringConstants.MenuNavigation);

Console.WriteLine(msg);

currentPos = Console.CursorTop;

min = currentPos;

max = menu.Length + min - 1;

PrintMenu();

SetActiveMenuItem();

ChangeColors(Background, Font);

ViewComponent.ClearWithotFirst();

return Choice;

}

}

}

## Clas ClassicMenu

using System;

namespace CourseWork.View

{

enum LastItem { Back, Exit}

class ClassicMenu : Menu

{

public ClassicMenu(LastItem line, ConsoleColor font, ConsoleColor background, params string [] menu)

:base(font, background, menu)

{

this.menu[this.menu.Length - 1] = (line == LastItem.Back)

? StringConstants.MenuLastItemBack

: StringConstants.MenuLastItemExit;

}

public int DoMenuAction()

{

return base.DoMenuAction(new string(' ', 1));

}

public int DoMenuActionMessage(string msg)

{

return base.DoMenuAction(msg);

}

}

}

## Class YesNoMenu

using System;

namespace CourseWork.View

{

class YesNoMenu : Menu

{

public YesNoMenu(ConsoleColor font, ConsoleColor background)

:

base(font, background, StringConstants.MenuYes,

StringConstants.MenuNo) { }

public new bool Choice

{

get

{

if (base.Choice == 0)

return true;

return false;

}

}

public new bool DoMenuAction(string question)

{

base.DoMenuAction(question);

return Choice;

}

}

}

## Class ViewComponent

using System;

using System.Text;

namespace CourseWork.View

{

class ViewComponent

{

public readonly static ConsoleColor Background = ConsoleColor.Black;

public readonly static ConsoleColor Font = ConsoleColor.Gray;

public readonly static ConsoleColor StatusBar = ConsoleColor.DarkGreen;

private static readonly string CleanString = new string(' ', Console.WindowWidth);

private readonly ClassicMenu mainMenu;

private readonly ClassicMenu searchChoice;

private readonly YesNoMenu yesNo;

internal ClassicMenu MainMenu

{

get

{

return mainMenu;

}

}

internal YesNoMenu YesNo

{

get

{

return yesNo;

}

}

internal ClassicMenu SearchChoice

{

get

{

return searchChoice;

}

}

internal ClassicMenu StatisticChoice

{

get; private set;

}

public ViewComponent()

{

mainMenu = new ClassicMenu(LastItem.Exit, Font, Background, StringConstants.InputData,

StringConstants.PrintTableData, StringConstants.WriteTextFile,

StringConstants.WriteBinaryFile, StringConstants.ReadTextFile,

StringConstants.ReadBinaryFile, StringConstants.SearchByField,

StringConstants.StatisticFunc);

searchChoice = new ClassicMenu(LastItem.Back, Font,

Background, StringConstants.SearchByGoodsName,

StringConstants.SearchByPrice);

yesNo = new YesNoMenu(Font, Background);

StatisticChoice = new ClassicMenu(LastItem.Back, Font, Background,

StringConstants.StatisticFuncMax, StringConstants.StatisticFuncMin);

}

public static void ClearStringTo(int top, int left)

{

Console.CursorTop = top;

Console.CursorLeft = 0;

Console.Write(new string(' ', left));

Console.CursorLeft = 0;

}

public static void ClearStringFrom(int top, int left)

{

Console.CursorTop = top;

Console.CursorLeft = left;

Console.Write(new string(' ', Console.WindowWidth - left));

Console.CursorTop--;

Console.CursorLeft = left;

}

public static void ClearString(int top)

{

ClearStringFrom(top, 0);

}

public static void ClearTo(int top, int left)

{

int prevTop = Console.CursorTop;

int prevLeft = Console.CursorLeft;

if (Console.CursorTop == top)

{

if (Console.CursorLeft == left)

{

return;

}

else if (Console.CursorLeft > left)

{

Console.CursorLeft = left;

left = prevLeft;

}

}

else if (Console.CursorTop > top)

{

Console.CursorTop = top;

Console.CursorLeft = left;

top = prevTop;

left = prevLeft;

}

ClearStringFrom(Console.CursorTop, Console.CursorLeft);

for (Console.CursorTop++; Console.CursorTop < top; ++Console.CursorTop)

{

ClearString(Console.CursorTop);

}

ClearStringTo(Console.CursorTop, left);

}

public static void ClearWithotFirst()

{

Console.CursorTop = 1;

for (int i = 1; i < Console.WindowHeight - 1; i++)

{

Console.Write(CleanString);

}

Console.CursorTop = 1;

}

}

}

# controller

## Class ControllerComponent

using CourseWork.Model;

using CourseWork.View;

using System;

using System.Collections.Generic;

using System.IO;

using System.Text.RegularExpressions;

namespace CourseWork.Controller

{

class ControllerComponent : IObserver

{

// public delegate void Statistcs();

private ViewComponent view;

private Balance model;

public ControllerComponent(ViewComponent view, Balance model)

{

this.model = model;

this.view = view;

}

public void DoActions()

{

model.Send += Update;

model.Notify();

while (view.MainMenu.DoMenuAction() != Menu.ExitCode)

{

switch (view.MainMenu.Choice)

{

case 0: ConsoleReadData(); break;

case 1: PrintTableModel(model); break;

case 2: WriteTextFile(); break;

case 3: WriteBinaryFile(); break;

case 4: ReadFromTextFile(); break;

case 5: ReadFromBinaryFile(); break;

case 6: Search(); break;

case 7: StatisticFunc(); break;

}

}

}

#region Пошук

private void Search()

{

if (model.Length > 0)

{

view.SearchChoice.DoMenuAction();

switch (view.SearchChoice.Choice)

{

case 0: SearchByName(); break;

case 1: SearchByPrice(); break;

}

}

else

{

Console.WriteLine(StringConstants.EmptyList);

Console.ReadKey();

}

}

private void SearchByPrice()

{

Console.Write(StringConstants.GoodsPrice);

DailyBalance searched = model.Search(model.CompareByPrice,

ReadFromConsole.ReadDouble());

PrintResults(searched);

}

private void SearchByName()

{

Console.Write(StringConstants.GoodsName);

Balance searched = null;

try

{

searched = (model.Search(model.CompareByName,

ReadFromConsole.ReadStringAndReaction(

StringConstants.GoodsNameRegex,

StringConstants.OnlyCharactersAndNumbersError,

RegexOptions.IgnoreCase)));

}

finally

{

PrintResults(searched);

}

}

private void PrintResults(Balance searched)

{

if (searched == null || searched.Length > 0)

{ PrintTableModel(searched); }

else

{

Console.WriteLine(StringConstants.SearchNoFound);

Console.ReadKey();

}

}

private void PrintSimilarsCount(int lenghtBefore, int count)

{

if (lenghtBefore > 0 && (lenghtBefore + count) > model.Length)

{

Console.WriteLine(StringConstants.SimilarSales + (lenghtBefore + count - model.Length).ToString());

}

}

private void ReadFromTextFile()

{

List<Sale> balance;

bool IsWasException = true;

try

{

balance = Balance.ReadFromTextFile(StringConstants.DefaultTextFileName);

int count = model.Length;

foreach (Sale element in balance)

{

model.Add(element);

}

Console.WriteLine(StringConstants.ReadAccept + StringConstants.DefaultTextFileName);

PrintSimilarsCount(count, balance.Count);

PrintTableModel(model);

IsWasException = false;

}

catch (FileNotFoundException)

{

Console.WriteLine(StringConstants.FileNotFound);

}

catch(FormatException)

{

Console.WriteLine(StringConstants.FormatError);

}

catch(ArgumentOutOfRangeException)

{

Console.WriteLine(StringConstants.ArgumentOutError);

}

finally

{

if (IsWasException)

{

Console.ReadKey();

}

}

}

private void ReadFromBinaryFile()

{

List<Sale> read;

try

{

read = Balance.ReadFromBinaryFile(StringConstants.DefaultBinaryFileName);

int count = model.Length;

foreach(Sale element in read)

{

model.Add(element);

}

Console.WriteLine(StringConstants.ReadAccept + StringConstants.DefaultBinaryFileName);

PrintSimilarsCount(count, read.Count);

PrintTableModel(model);

}

catch (FileNotFoundException)

{

ReadFromConsole.WriteError(StringConstants.FileNotFound);

}

catch (IOException)

{

ReadFromConsole.WriteError(StringConstants.FileIOError);

}

}

private void WriteBinaryFile()

{

if (model.Length > 0)

{

try

{

model.WriteToBinaryFile(StringConstants.DefaultBinaryFileName);

Console.WriteLine(StringConstants.WriteAccept + StringConstants.DefaultBinaryFileName);

}

catch (IOException)

{

ReadFromConsole.WriteError(StringConstants.FileIOErrorOpen);

}

catch (UnauthorizedAccessException)

{

ReadFromConsole.WriteError(StringConstants.AccessException);

}

}

else

{ Console.WriteLine(StringConstants.EmptyList); }

Console.ReadKey();

}

private void WriteTextFile()

{

if (model.Length > 0)

{

try

{

model.WriteToTextFile(StringConstants.DefaultTextFileName);

Console.WriteLine(StringConstants.WriteAccept

+ StringConstants.DefaultTextFileName);

Console.ReadKey();

}

catch (IOException)

{

ReadFromConsole.WriteError(StringConstants.FileIOError);

}

catch(UnauthorizedAccessException)

{

ReadFromConsole.WriteError(StringConstants.AccessException);

}

}

else

{

ReadFromConsole.WriteError(StringConstants.EmptyList);

}

}

public void ConsoleReadData()

{

do

{

model.Add(Input());

} while (view.YesNo.DoMenuAction(StringConstants.QuestionAddSell));

}

public Sale Input()

{

Seller seller = InputSeller();

Date saleDate = InputDate(StringConstants.DaySell, StringConstants.MonthSell);

Goods goods = InputGoods();

Console.Write(StringConstants.Amount);

double amount = ReadFromConsole.ReadDouble(StringConstants.PositiveNumRegex);

return new Sale(seller, goods, saleDate, amount);

}

public Goods InputGoods()

{

Date receiving = InputDate(StringConstants.GoodsReceiptDay, StringConstants.GoodsReceiptMonth);

Console.Write(StringConstants.GoodsName);

string name = ReadFromConsole.ReadStringAndReaction(

StringConstants.GoodsNameRegex,

StringConstants.OnlyCharactersAndNumbersError,

RegexOptions.IgnoreCase);

Console.Write(StringConstants.GoodsPrice);

double price = ReadFromConsole.ReadDouble(StringConstants.PositiveNumRegex);

Console.Write(StringConstants.GoodsType);

GoodsType type = UpDownChoice();

return new Goods(name, price, receiving, type);

}

public Seller InputSeller()

{

Date recruited = InputDate(StringConstants.SellerEmployeDay, StringConstants.SellerEmployeMonth);

Console.Write(StringConstants.SellerName);

string name = ReadFromConsole.ReadStringAndReaction(

StringConstants.NameSurnameRegex,

StringConstants.OnlyCharactersError,

RegexOptions.IgnoreCase);

Console.Write(StringConstants.SellerSurname);

string surname = ReadFromConsole.ReadStringAndReaction(

StringConstants.NameSurnameRegex,

StringConstants.OnlyCharactersError,

RegexOptions.IgnoreCase);

return new Seller(name, surname, recruited);

}

public Date InputDate(string dayMessage, string monthMessge)

{

int day;

int month;

Console.Write(monthMessge);

month = UpDownIntInput(Date.MinMonth, Date.MaxMonth);

Console.Write(dayMessage);

day = UpDownIntInput(Date.MinDay, Date.MaxDayMonth(month));

return new Date(day, month);

}

/// <returns>Введене число</returns>

public int UpDownIntInput(int min, int max)

{

if (max <= min)

{ throw new ArgumentOutOfRangeException(); }

int currentNum = min;

int left = Console.CursorLeft;

int top = Console.CursorTop;

int numbers = 0;

for (int i = max; i > 0; i /= 10)

{ numbers++; }

string format = "{0," + numbers + ":" + new string('0', numbers) + '}';

Console.CursorVisible = false;

Console.Write(format, min);

ConsoleKeyInfo pressedKey;

do

{

pressedKey = Console.ReadKey(true);

switch (pressedKey.Key)

{

case ConsoleKey.DownArrow:

if (currentNum > min)

{ currentNum--; }

else

{ currentNum = max; }

break;

case ConsoleKey.UpArrow:

if (currentNum < max)

{ currentNum++; }

else

{ currentNum = min; }

break;

}

Console.CursorLeft -= numbers;

Console.Write(format, currentNum);

} while (pressedKey.Key != ConsoleKey.Enter);

Console.WriteLine();

return currentNum;

}

public GoodsType UpDownChoice()

{

ConsoleKeyInfo pressed;

GoodsType choice = GoodsType.Unpacked;

Console.Write(StringConstants.GoodsTypeUnpacked);

do

{

pressed = Console.ReadKey(true);

if (pressed.Key == ConsoleKey.UpArrow || pressed.Key == ConsoleKey.DownArrow)

{

if (choice == GoodsType.Unpacked)

{

choice = GoodsType.WeighedOut;

ViewComponent.ClearStringFrom(Console.CursorTop,

Console.CursorLeft - StringConstants.GoodsTypeUnpacked.Length);

Console.Write(StringConstants.GoodsTypeWeighedOut);

}

else

{

choice = GoodsType.Unpacked;

ViewComponent.ClearStringFrom(Console.CursorTop,

Console.CursorLeft - StringConstants.GoodsTypeWeighedOut.Length);

Console.Write(StringConstants.GoodsTypeUnpacked);

}

}

} while (pressed.Key != ConsoleKey.Enter);

Console.WriteLine();

return choice;

}

#endregion

public void PrintTableModel(Balance balance)

{

if (balance.Length == 0)

{

Console.WriteLine(StringConstants.EmptyList);

Console.ReadKey();

return;

}

Console.Write(StringConstants.TopLine);

Console.Write(StringConstants.HeaderFormat,

StringConstants.HeaderDate, StringConstants.HeaderGoodsName,

StringConstants.HeaderGoodsType, StringConstants.HeaderPrice,

StringConstants.HeaderAmount, StringConstants.HeaderTotalPrice,

StringConstants.HeaderSurname);

Console.Write(StringConstants.CenterLine);

Console.Write(balance.ToTableRow());

Console.Write(StringConstants.BottomLine);

Console.ReadKey();

}

public void StatisticFunc()

{

view.StatisticChoice.DoMenuAction();

switch (view.StatisticChoice.Choice)

{

case 0: PrintStatistic(model.Max); break;

case 1: PrintStatistic(model.Min); break;

}

}

public void PrintStatistic(Balance.Statistics statistic)

{

try

{

Console.Write(StringConstants.TopLine);

Console.Write(StringConstants.HeaderFormat,

StringConstants.HeaderDate, StringConstants.HeaderGoodsName,

StringConstants.HeaderGoodsType, StringConstants.HeaderPrice,

StringConstants.HeaderAmount, StringConstants.HeaderTotalPrice,

StringConstants.HeaderSurname);

Console.Write(StringConstants.CenterLine);

Console.Write(statistic().ToTableRow());

Console.Write(StringConstants.BottomLine);

}

catch(Exception)

{

ViewComponent.ClearWithotFirst();

Console.WriteLine(StringConstants.EmptyList);

}

finally

{

Console.ReadKey();

}

}

public void Update()

{

Console.CursorTop = 0;

Console.BackgroundColor = ViewComponent.StatusBar;

Console.Write(StringConstants.StatusBarSellsFormat, StringConstants.StatusBarSells, model.Length);

Console.Write(StringConstants.StatusBarCashFormat, StringConstants.StatusBarCash, model.TotalCash);

Console.WriteLine();

Console.BackgroundColor = ViewComponent.Background;

}

}

}

## Class ReadFromConsole

using CourseWork.View;

using System;

using System.Text.RegularExpressions;

namespace CourseWork.Controller

{

static class ReadFromConsole

{

static int left;

static int top;

static int count;

static Match matcher;

public static double ReadDouble(string pattern)

{

double num = 0;

bool IsParseAccepted = false;

do

{

try

{

num = Double.Parse(ReadStringAndReaction(pattern, StringConstants.DoubleParseError, RegexOptions.IgnoreCase));

IsParseAccepted = true;

}

catch

{

WriteError(StringConstants.DoubleParseError);

}

} while (!IsParseAccepted);

return num;

}

public static void WriteError(string message)

{

Console.Write(message);

Console.ReadKey(true);

ViewComponent.ClearTo(top, left);

Console.CursorTop = top;

Console.CursorLeft = left;

}

private static string ReadString()

{

string line;

left = Console.CursorLeft;

top = Console.CursorTop;

line = Console.ReadLine();

count = line.Length;

return line;

}

public static string ReadStringAndReaction(string pattern, string errorMessage, RegexOptions options)

{

string line;

do

{

line = ReadString();

matcher = Regex.Match(line, pattern, options);

if (!matcher.Success)

{

WriteError(errorMessage);

}

} while (!matcher.Success);

return line;

}

}

}

# pattern Observer

## Interface IObservable

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseWork

{

public delegate void NotifyObserver();

interface IObservable

{

event NotifyObserver Send;

void Notify();

}

}

## Interface IObserver

namespace CourseWork

{

interface IObserver

{

void Update();

}

}