

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Харківський національний університет радіоелектроніки

Кафедра ЕОМ

ЛАБОРАТОРНА РОБОТА №5

MULTI-THREADING .NET

Виконав:

ст. гр. КІУКІ-20-5

Мавринський О.Д.

Прийняла:

Ляшова А.О.

Харків 2022

## 1.1 Мета роботи

Вивчення можливостей розробки багатопоточних додатків мовою С#, а також дослідження механізмів синхронізації потоків в .NET..

## 1.2 Хід роботи

Виконання завдання передбачається використання технології Windows Forms або WPF.

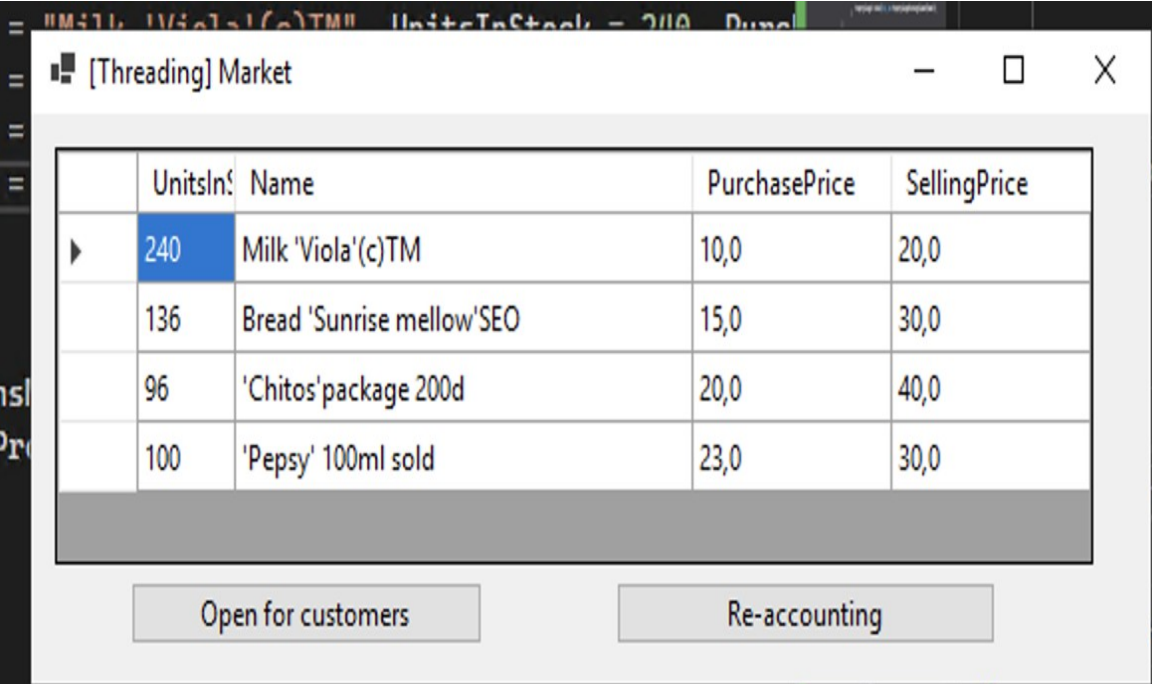
Додаток моделює роботу магазину, який має список товарів на складі.

Кожен вид товару в магазині має назву, кількість одиниць на складі, закупівельну ціну та ціну продажу покупцям.

Магазин працює у двох режимах:

- магазин відкрит для покупців - при цьому в окремому потоці відбувається генерація випадкової кількості покупців, кожен з яких купує різну кількість товарів, які видаляються зі складу магазину, а також сплачує відповідну суму до каси магазину.
- магазин знаходиться на переобліку - в цей момент магазин у окремому потоці на протязі вказаного часу поповнює склад товарами та сплачує їх закупівельні ціни.

Забезпечити динамічне формування логів дій покупців та переобліку магазину у файл XML-формату.



UnitsInStock	Name	PurchasePrice	SellingPrice
240	Milk 'Viola'(c)TM	10,0	20,0
136	Bread 'Sunrise mellow'SEO	15,0	30,0
96	'Chitos'package 200d	20,0	40,0
100	'Pepsi' 100ml sold	23,0	30,0

Open for customers      Re-accounting

App.cs

---

```
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Xml.Serialization;
namespace lb5 {
    public partial class Form1 : Form{
        private void SerializeToXml<T>(T obj) {
            XmlSerializer serializer = new XmlSerializer(typeof(T));
            using (TextWriter writer = new StreamWriter("datatable.xml", true))
            { serializer.Serialize(writer, obj);} }
        public Form1() {
            InitializeComponent();
            _products = new List<ProductModel> {
                new ProductModel { Name = "Milk 'Viola'(c)TM", UnitsInStock = 240, PurchasePrice
= 10.0m, SellingPrice = 20.0m },
                new ProductModel { Name = "Bread 'Sunrise mellow'SEO", UnitsInStock = 136,
PurchasePrice = 15.0m, SellingPrice = 30.0m },
                new ProductModel { Name = "'Chitos'package 200d", UnitsInStock = 96,
PurchasePrice = 20.0m, SellingPrice = 40.0m },
                new ProductModel { Name = "'Pepsi' 100ml sold", UnitsInStock = 100,
PurchasePrice = 23.0m, SellingPrice = 30.0m }};
            _random = new Random();
            dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
            dataGridView1.DataSource = Products; }

        public class ProductModel {
            public event PropertyChangedEventHandler PropertyChanged;
            private int _unitsInStock;
            public int UnitsInStock{
                get { return _unitsInStock; }
                set{if (_unitsInStock != value){ _unitsInStock = value;
                    PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(UnitsInStock)));} }}
            private string _name;
            public string Name {
                get { return _name; }
                set {if (_name != value) { _name = value;
                    PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(Name)));} }}
            private decimal _purchasePrice;
            public decimal PurchasePrice{
                get { return _purchasePrice; }
```

```

        set {
            if (_purchasePrice != value) {
                _purchasePrice = value;
                PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(PurchasePrice))); } } }
        private decimal _sellingPrice;
        public decimal SellingPrice {
            get { return _sellingPrice; }
            set {
                if (_sellingPrice != value) {
                    _sellingPrice = value;
                    PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(SellingPrice))); } } }
    public class CustomerAction {
        public string ProductName;
        public int UnitsPurchased;
        public int UnitsInStock;
        public decimal AmountPaid;
        public decimal Casa;}
    public class ReAccountingAction{
        public string ProductName;
        public int UnitsReplenished;
        public int UnitsInStock;
        public decimal AmountPaid;
        public decimal Casa;}
    private List<ProductModel> _products;
    private decimal _money = 0.0m;
    private Random _random;
    public ObservableCollection<ProductModel> Products{
        get { return new ObservableCollection<ProductModel>(_products); } }
    public event PropertyChangedEventHandler PropertyChanged;
    public void OnPropertyChanged(string property){
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(property));}
    private void OpenForCustomersModeButton_Click(object sender, EventArgs e){
        Thread customerThread = new Thread(() =>{
            for (int i = 0; i < _random.Next(1, 5); i++){
                int productIndex = _random.Next(0, _products.Count);
                ProductModel selectedProduct = _products[productIndex];
                int numUnitsToPurchase = _random.Next(1, selectedProduct.UnitsInStock + 1);
                decimal amountPaid = numUnitsToPurchase * selectedProduct.SellingPrice;

```

```

        if (selectedProduct.UnitsInStock - numUnitsToPurchase > 0) {
            selectedProduct.UnitsInStock -= numUnitsToPurchase;
            _money += amountPaid;
            CustomerAction customerAction = new CustomerAction {
                ProductName = selectedProduct.Name,
                UnitsPurchased = numUnitsToPurchase,
                UnitsInStock = selectedProduct.UnitsInStock,
                AmountPaid = amountPaid,
                Casa = _money };
            try{ SerializeToXml(customerAction); }
            catch (IOException ex){
                MessageBox.Show($"Failed to serialize customer action: {ex.Message}"); }}
            Thread.Sleep(_random.Next(1000, 2000));
        });
        customerThread.Start(); }

private void ReAccountingModeButton_Click(object sender, EventArgs e) {
    Thread reAccountingThread = new Thread(() => {
        for (int i = 0; i < _random.Next(1, 5); i++) {
            int productIndex = _random.Next(0, _products.Count);
            ProductModel selectedProduct = _products[productIndex];
            int numUnitsToPurchase = _random.Next(10, 20);
            decimal amountPaid = numUnitsToPurchase * selectedProduct.PurchasePrice;
            if (_money - amountPaid > 0){
                selectedProduct.UnitsInStock += numUnitsToPurchase;
                _money -= amountPaid;
                ReAccountingAction reAccountingAction = new ReAccountingAction {
                    ProductName = selectedProduct.Name,
                    UnitsReplenished = numUnitsToPurchase,
                    UnitsInStock = selectedProduct.UnitsInStock,
                    AmountPaid = amountPaid,
                    Casa = _money
                };
                try{
                    OnPropertyChanged(nameof(_products));
                    SerializeToXml(reAccountingAction); }
                catch (IOException ex){
                    MessageBox.Show($"Failed to serialize re-accounting action {ex.Message}"); }}
                Thread.Sleep(_random.Next(1000, 2000));}});
        reAccountingThread.Start(); }}}

```



### 1.3 ВИСНОВКИ