МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**НАЦІОНАЛЬНИЙ ЛІСОТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

Кафедра інформаційних технологій

**КУРСОВИЙ ПРОЕКТ**

|  |  |
| --- | --- |
| з дисципліни | Технології розподілених систем та паралельних обчислень |
|  |  |
| на тему: | **Задача про біржу** |
|  |

Студента 3 курсу групи КН-31

Напряму підготовки 122 – “Комп’ютерні науки”

Манжула Д. В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник: ***завідувач кафедри ІТ,\_ професор, д.т.н.\_Соколовський Я.І.\_\_ \_***

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_ Оцінка: ECTS \_\_\_\_

Члени комісії:

|  |  |  |
| --- | --- | --- |
|  |  | ***Дендюк М. В.*** |
| (підпис) |  | (прізвище та ініціали) |
|  |  | ***Яркун В.І.*** |
| (підпис) |  | (прізвище та ініціали) |
|  |  | ***Нечепуренко А.В.*** |
| (підпис) |  | (прізвище та ініціали) |
| (підпис) |  | (прізвище та ініціали) |

Львів – 2019

Зміст

[Вступ 3](#_Toc388524198)

[1. Технічне завдання 4](#_Toc388524199)

[2. Середовище та технології реалізації системи 5](#_Toc388524200)

[**2.1. Поняття багато потоковості 5**](#_Toc388524201)

[**2.2. Клас Thread 7**](#_Toc388524202)

[**2.3. Поняття черги 9**](#_Toc388524203)

[3. Програмна реалізація системи 10](#_Toc388524204)

[**3.1. Алгоритм завдання 10**](#_Toc388524205)

[**3.2. Проектування класів 11**](#_Toc388524206)

[**3.3. Графічний інтерфейс 12**](#_Toc388524208)

[4. Результат моделювання 13](#_Toc388524209)

[Висновок 14](#_Toc388524210)

[Список використаної літератури 15](#_Toc388524211)

[Додатки 16](#_Toc388524212)

# Вступ

Використання сучасної техніки і сучасних технологій паралельного програмування дало можливість розв’язувати як класичні так і надскладні задачі при малих затратах часу на розробку. Програмні продукти алгоритм яких реалізований за допомогою паралельних технологій працюють продуктивніше, якісніше, швидше економлячи при цьому енергетичні ресурси.

Одним з ключових аспектів у сучасному програмуванні є багатопоковість. За допомогою багатопоковості ми можемо виділити в додатку декілька потоків, які будуть виконувати різні завдання одночасно.

В даній роботі була розроблена багатопотокова програма для моделювання роботи біржі. Програма реалізована на мові C# з використанням бібліотеки System.Threading.

# 1. Технічне завдання

На торгах брокери продають акції декількох фірм. На біржі здійснюються операції купівлі/продажу акцій. В залежності від кількості проданих акцій їх ціни змінюються (чим більше проданих акцій, тим вища стає ціна). Брокери пропонують на продаж деяку кількість акцій, ціна яких переглядається кожні 30 секунд. Від активності і зростання/падіння котировок акцій (ціни, яку покупці готові заплатити за акції) змінюється індекс біржі (зростає або падає). Біржа може призупинити торги при різкому падінні індексу біржі. Розробити багато потоковий додаток, що моделює роботу біржі.

# 2. Середовище та технології реалізації системи

## 2.1. Поняття багато потоковості

Однопотоковий додаток схожий на супермаркет, де працює єдиний касир. Один касир обходиться власнику магазину дешевше, але він може обслужити за певний час порівняно небагато клієнтів. Але як тільки супермаркет оголошує розпродаж, черга біля каси зростає. При цьому деякі клієнти виявляють сильне невдоволення. Тут ми маємо стандартний сценарій з "вузьким місцем": занадто багато даних і занадто маленька пропускна здатність. Словом, потрібні додаткові касири.

Так само відбувається з потоками і додатками. Потік дозволяє додаткам розділяти задачі і працювати над кожною незалежно, щоб максимально ефективно задіяти процесор і призначене для користувача час.

Потік являє собою координовану одиницю виконуваного коду. Своїм походженням цей термін зобов'язаний поняттю "потік виконання". При організації багатозадачності на основі потоків у кожного процесу повинен бути принаймні один потік, хоча їх може бути і більше. Це означає, що в одній програмі одночасно можуть вирішуватися два завдання і більше. Наприклад, текст може форматуватися в редакторі тексту одночасно з його висновком на друк, за умови, що обидва ці дії виконуються в двох окремих потоках.

Відмінності в багатозадачності на основі процесів і потоків можуть бути зведені до наступного: багатозадачність на основі процесів організується для паралельного виконання програм, а багатозадачність на основі потоків - для паралельного виконання окремих частин однієї програми.

Головна перевага багатопотокової обробки полягає в тому, що вона дозволяє писати програми, які працюють дуже ефективно завдяки можливості вигідно використовувати час простою, неминуче виникає в ході виконання більшості програм. Як відомо, більшість пристроїв введення-виведення, будь то пристрої, підключені до мережевих портів, накопичувачі на дисках або клавіатура, працюють набагато повільніше, ніж центральний процесор (ЦП). Тому більшу частину свого часу програмі доводиться очікувати відправки даних на пристрій введення-виведення або прийому інформації з нього. А завдяки багатопотокової обробці програма може вирішувати якусь іншу задачу під час вимушеного простою.

Всі процеси складаються хоча б з одного потоку, який зазвичай називають основним, оскільки саме з нього починається виконання програми. З основного потоку можна створити інші потоки.

У мові С# і середовищі .NET Framework підтримуються обидва різновиди багатозадачності: на основі процесів і на основі потоків. Тому засобами С# можна створювати як процеси, так і потоки, а також управляти і тими і іншими. Для того щоб почати новий процес, від програмуючого потрібно зовсім небагато зусиль, оскільки кожен попередній процес абсолютно відокремлений від наступного.

Набагато більш важливою виявляється підтримка в С# багатопотокової обробки, завдяки якій спрощується написання високопродуктивних, багатопоточних програм на С# в порівнянні з деякими іншими мовами програмування.

Класи, які підтримують багатопоточне програмування, визначені в просторі імен System.Threading. Тому будь-яка багатопотокова програма на С# включає в себе наступний рядок коду:

using System.Threading;

Простір імен System.Threading містить різні типи, що дозволяють створювати багатопотокові програми. Мабуть, головним серед них є клас Thread, оскільки він представляє окремий потік.

## 2.2. Клас Thread

Основний функціонал для використання потоків у додатку зосереджений в просторі імен System.Threading. У ньому визначено клас, що представляє окремий потік - клас Thread.

Клас Thread визначає ряд методів і властивостей, які дозволяють управляти потоком і отримувати інформацію про нього. Основні властивості класу:

* Статична властивість CurrentContext дозволяє отримати контекст, в якому виконується потік
* Статична властивість CurrentThread повертає посилання на виконуваний потік
* Властивість IsAlive вказує, чи працює потік в поточний момент
* Властивість IsBackground вказує, чи є потік фоновим
* Властивість Name містить ім'я потоку
* Властивість Priority зберігає пріоритет потоку - значення перерахування ThreadPriority
* Властивість ThreadState повертає стан потоку - одне із значень перерахування ThreadState
* Деякі методи класу Thread:
* Статичний метод GetDomain повертає посилання домен програми
* Статичний метод GetDomainId повертає id домену програми, в якому виконується поточний потік
* Статичний метод Sleep зупиняє потік на певну кількість мілісекунд
* Метод Abort повідомляє середу CLR про те, що треба припинити потік, однак припинення роботи потоку відбувається не відразу, а тільки тоді, коли це стає можливо. Для перевірки завершеності потоку слід опитувати його властивість ThreadState
* Метод Interrupt перериває потік на деякий час
* Метод Join блокує виконання викликав його потоку до тих пір, поки не завершиться потік, для якого був викликаний даний метод
* Метод Resume відновлює роботу раніше припиненого потоку
* Метод Start запускає потік
* Метод Suspend призупиняє потік

СТАТУС ПОТОКУ

Статуси потоку містяться в перерахуванні ThreadState :

* Aborted : потік зупинений, але поки ще остаточно не завершений
* AbortRequested : для потоку викликаний метод Abort, але зупинка потоку ще не відбулася
* Background : потік виконується у фоновому режимі
* Running : потік запущений і працює ( не зупинено )
* Stopped : потік завершено
* StopRequested : потік отримав запит на зупинку
* Suspended : потік призупинений
* SuspendRequested : потік отримав запит на призупинення
* Unstarted : потік ще не був запущений
* WaitSleepJoin : потік заблокований в результаті дії методів Sleep або Join

У процесі роботи потоку його статус багаторазово може змінитися під дією методів. Так, на самому початку ще до застосування методу Start його статус має значення Unstarted. Запустивши потік, ми змінимо його статус на Running. Викликавши метод Sleep, статус зміниться на WaitSleepJoin. А застосовуючи метод Abort, ми тим самим переведемо потік в стан AbortRequested, а потім Aborted, після чого потік остаточно завершиться.

Пріоритети потік розташовуються в перерахуванні ThreadPriority :

Lowest, BelowNormal, Normal, AboveNormal, Highest.

За замовчуванням потоку задається значення Normal. Однак ми можемо змінити пріоритет у процесі роботи програми. Наприклад, підвищити важливість потоку, встановивши пріоритет Highest. CLR буде зчитувати і аналізувати значення пріоритету і на їх підставі виділяти даному потоку або інша кількість часу.

## 2.3. Поняття черги

Черги корисні для зберігання повідомлення в тому порядку, в якому вони були отримані, для подальшої послідовної обробки цих повідомлень. Цей клас реалізує чергу у вигляді циклічного масиву. Об'єкти, збережені в колекції Queue, вставляються в один кінець черги, а видаляються з іншого кінця.

Ємність колекції Queue - це кількість елементів, яке може вмістити колекція Queue. Коли в колекцію Queue додаються елементи, її ємність автоматично збільшується належним чином за допомогою перерозподілу. Ємність може бути зменшена за допомогою виклику методу TrimToSize.

Коефіцієнт зростання - це число, на яке множиться поточна ємність при необхідності її збільшення. Коефіцієнт зростання визначається при побудові колекції Queue. За замовчуванням коефіцієнт зростання дорівнює 2,0. Ємність колекції Queue завжди збільшується не менше, ніж на 4, незалежно від коефіцієнта зростання. Наприклад, кожен раз, коли потрібно збільшити ємність колекції Queue з коефіцієнтом зростання, рівним 1,0, ця ємність завжди збільшуватися на чотири.

У колекції Queue значення null є допустимим. Також дозволяється повторення елементів.

Про універсальної версії цієї колекції см. System.Collections.Generic.Queue <T>.

# 3. Програмна реалізація системи

## 3.1. Алгоритм завдання

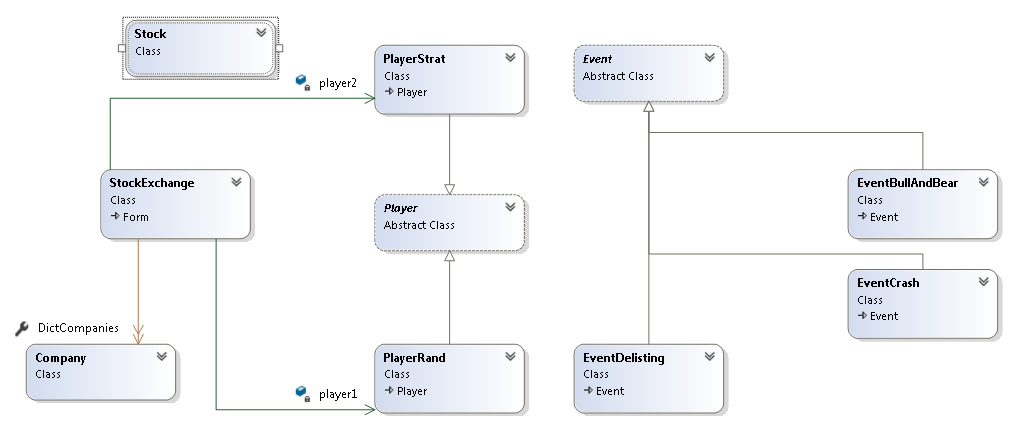
Існує масив брокерів, які продають акції декількох фірм. З деяким інтервалом запускається потік, який виконує операції купівлі/продажу акцій. В розробленій програмі в залежності від кількості проданих акцій їх ціни змінюються. Брокери пропонують на продаж деяку кількість акцій, ціна яких переглядається кожні 30 секунд. Від активності і зростання/падіння котировок акцій (ціни, яку покупці готові заплатити за акції) змінюється індекс біржі (зростає або падає). Біржа може призупинити торги при різкому падінні індексу біржі. Зміна даних відбувається в критичній секції з допомогою оператора ***lock()***.

Програма виконується циклічно, весь час в окремому потоці оновлюючи дані таблиці головного вікна. Робота моделюється поки користувач не закриє головне вікно програми. Робота програми виконується чітко за визначеною послідовністю.

Праве частина головної форми показує стан біржі, який змінюється у певних одиницях модельного часу (представлено днями). Також є можливість призупинити виконання та перезапустити відповідними кнопка головної форми. Бізнес логіка купівлі і продажу акцій винесена у спеціальний клас брокера.

## 3.2. Проектування класів

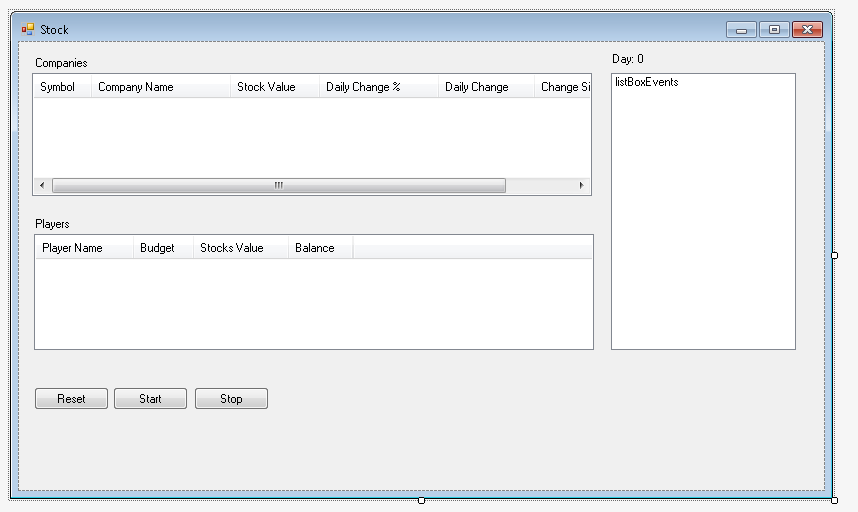
В процесі проектування системи було розроблено наступні класи представлені на рисунку 2.



*Рис.2. Класи запроектовані в системі*

## 3.3. Графічний інтерфейс

Розроблення програмного додатку реалізовувалось шляхом створення проекту «Windows Forms Application». На формі розташовані елементи керування діалогом. Розроблена форма представлена на Рис.3.



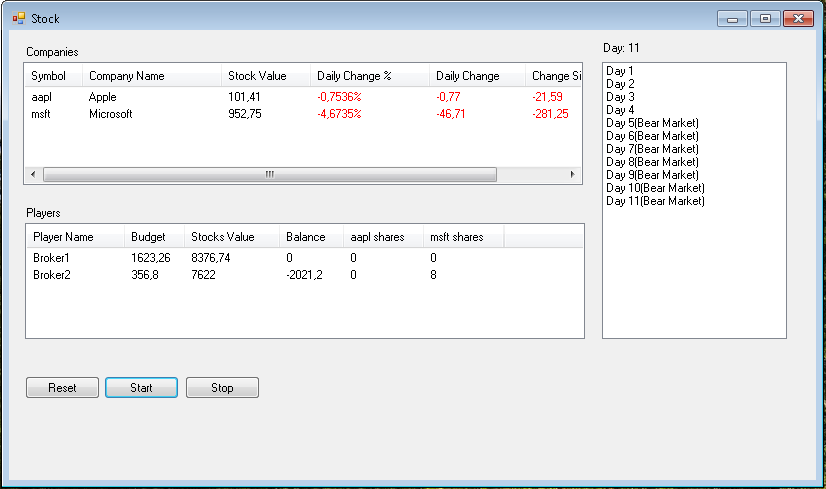
*Рис. 3. Вигляд форми додатку*

Результат моделювання відображається динамічно по мірі події оновлення таймеру, що виконується в фоновому потоці. Запуск таймерів розпочинається одразу після відображення форми, а завершення роботи таймерів, після закриття головного вікна.

# 4. Результат моделювання

Процес відображення результатів відбувається відповідно до роботи додатку та запущених у ньому потоків. На формі виводитимуться повідомлення що відображають стан біржі.

Результат виконання багато потокового додатку можна побачити на рисунках нижче.



*Рис. 4. Результат виконання багато потокового додатку*

# Висновок

У курсовій роботі розроблено багато-потокову програму, що моделює процес роботи біржі. Для реалізації багатопотоковості були використані класи із System.Threading. Спостерігати за роботою системи в процесі моделювання можна на формі, де явно відображаються всі дії.

У зв’язку з роботою із багатьма потоками програма забезпечує синхронізацію потоків так, що потоки працюють лише тоді коли в черзі є завдання, а також із обмеженим доступом до методів черги.

# Список використаної літератури

1. The Little Book of Semaphores, - Second Edition, - Copyright 2005 Allen B. Downey.
2. Богачев К.Ю. Основы параллельного программирования. – М.:

БИНОМ. Лаб. знаний, 2003. – 342 с

1. D. Y. Cheng. A survey of parallel programming languages and tools. Moffett Field, 1993.
2. Ендрю Троелсен. [Мова програмування C# 5.0 і платформа.NET 4.5 (видання 6-е)](http://uastream.net/20070-mova-programuvannya-c-50-i-platforma-net-45-vidannya-6-e.html). Вільямс, 2013.
3. Gaston C. Hillar. Professional Parallel Programming with C#: Master Parallel Extensions with.NET 4., Willey Publishing Inc., 2012.
4. [http://msdn.microsoft.com/ru-ru/library/z1zx9t92(v=VS.90).aspx](http://msdn.microsoft.com/ru-ru/library/z1zx9t92%28v=VS.90%29.aspx)
5. Соколовський Я.І., Яркун В.І. Методичні вказівки до виконання курсового проектування з дисципліни «Технології розподілених та паралельних обчислень» для студентів денної та заочної форми навчання за напрямом «Комп’ютерні науки», НЛТУ України. Львів. 2013р.

# Додатки

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

using System.Threading;

namespace StockExchangeDraft

{

public partial class StockExchange : Form

{

public StockExchange()

{

InitializeComponent();

}

object locker = new object();

//creating the objects

private static int count = 0;

private static double bullAndBearCheck = 0;

private static Company apple = new Company("Apple", "aapl", 123, 1);

private static Company microsoft = new Company("Microsoft", "msft", 1234, 1);

private static PlayerRand player1 = new PlayerRand("Broker1", 10000);

private static PlayerStrat player2 = new PlayerStrat("Broker2", 10000);

private static EventCrash crash = new EventCrash("Crash", 0.99, "Stock Market Crash happened on day ");

private static EventDelisting delisting = new EventDelisting();

private static EventBullAndBear bullAndBear = new EventBullAndBear(0.04);

//creating the lists

private static Dictionary<string, Company> dictCompanies = new Dictionary<string, Company>();

private static List<Player> lstPlayers = new List<Player>();

public static int Count { get => count; set => count = value; }

public static double BullAndBearCheck { get => bullAndBearCheck; set => bullAndBearCheck = value; }

public static Company Apple { get => apple; set => apple = value; }

public static Company Microsoft { get => microsoft; set => microsoft = value; }

public static PlayerRand Player1 { get => player1; set => player1 = value; }

public static PlayerStrat Player2 { get => player2; set => player2 = value; }

public static EventCrash Crash { get => crash; set => crash = value; }

public static EventDelisting Delisting { get => delisting; set => delisting = value; }

public static EventBullAndBear BullAndBear { get => bullAndBear; set => bullAndBear = value; }

public static List<Player> LstPlayers { get => lstPlayers; set => lstPlayers = value; }

public static Dictionary<string, Company> DictCompanies { get => dictCompanies; set => dictCompanies = value; }

//Changes label's color

public static void SetColorPositivity(ListViewItem.ListViewSubItem label, double checkNumber)

{

if (checkNumber > 0) label.ForeColor = Color.Green;

if (checkNumber < 0) label.ForeColor = Color.Red;

if (checkNumber == 0) label.ForeColor = Color.Blue;

}

public static void SetColorPositivity(Label label, double checkNumber)

{

if (checkNumber > 0) label.ForeColor = Color.Green;

if (checkNumber < 0) label.ForeColor = Color.Red;

if (checkNumber == 0) label.ForeColor = Color.Blue;

}

private static void CompaniesListViewInitialize(Dictionary<string, Company> dictCompanies, ListView stocksListView)

{

foreach (KeyValuePair<string, Company> item in dictCompanies)

{

Company company = item.Value;

ListViewItem listViewItem = new ListViewItem(company.CompanySymbol, 0);

listViewItem.SubItems.Add(company.CompanyName);

listViewItem.SubItems.Add(company.StockValue.ToString());

listViewItem.SubItems.Add(company.DailyChangePercentage.ToString());

listViewItem.SubItems.Add(company.DailyChangeUsd.ToString());

listViewItem.SubItems.Add(company.ChangeSinceStart.ToString());

//this property enables setting different colors for subitems

listViewItem.UseItemStyleForSubItems = false;

stocksListView.Items.Add(listViewItem);

company.ListViewItemCompany = listViewItem;

}

}

private static void CompaniesListViewDelisting(Company company, ListBox listBoxEvents)

{

ListViewItem listViewItem = company.ListViewItemCompany;

if(company.CompanyDelisted)

{

listBoxEvents.Items.Add(company.CompanyName + Delisting.EventMessageBoxText);

listViewItem.SubItems[1].Text = "Delisted";

listViewItem.SubItems[1].ForeColor = Color.Red;

for (int i = 2; i <= 5; i++)

{

listViewItem.SubItems[i].Text = "---";

listViewItem.SubItems[i].ForeColor = Color.Red;

}

}

}

private static void CompaniesListViewUpdate(Dictionary<string, Company> dictCompanies, ListBox listBoxEvents)

{

if (dictCompanies.Count <= 0)

return;

foreach (KeyValuePair<string, Company> item in dictCompanies)

{

Company company = item.Value;

ListViewItem listViewItem = company.ListViewItemCompany;

if (company.CompanyDelisted)

continue;

try

{

//starting indexing with 1, because 0 is a company symbol

listViewItem.SubItems[1].Text = company.CompanyName;

listViewItem.SubItems[2].Text = company.StockValue.ToString();

listViewItem.SubItems[3].Text = company.DailyChangePercentage.ToString() + "%";

SetColorPositivity(listViewItem.SubItems[3], company.DailyChangePercentage);

listViewItem.SubItems[4].Text = company.DailyChangeUsd.ToString();

SetColorPositivity(listViewItem.SubItems[4], company.DailyChangeUsd);

listViewItem.SubItems[5].Text = company.ChangeSinceStart.ToString();

SetColorPositivity(listViewItem.SubItems[5], company.ChangeSinceStart);

}

catch (IndexOutOfRangeException e)

{

throw new ArgumentOutOfRangeException("index parameter is out of range.", e);

}

}

}

private static void CompaniesListViewReset(ListView stocksListView)

{

foreach (ListViewItem listItem in stocksListView.Items)

{

stocksListView.Items.Remove(listItem);

}

}

//updates columns for all existing companies in playersListView

public static void PlayersListViewColumnsUpdate(Dictionary<string, Company> companies, ListView playersListView, List<Player> players)

{

foreach (KeyValuePair<string, Company> item in companies)

{

Company company = item.Value;

bool columnExists = false;

string companySharesText = company.CompanySymbol + " shares";

//check if the column already exists in the listView

foreach (ColumnHeader column in playersListView.Columns)

{

if (column.Text == companySharesText)

columnExists = true;

}

//If there are is no column like that, we want to add one

if (!columnExists)

{

playersListView.Columns.Add(companySharesText, 80, HorizontalAlignment.Left);

}

}

}

public static void PlayersListViewAddSubItems(Player player, ListView playersListView)

{

if (player.ListViewItemPlayer == null)

return;

//fills the SubItems with "0" in order to make their count equal to columns count

while(player.ListViewItemPlayer.SubItems.Count < playersListView.Columns.Count)

player.ListViewItemPlayer.SubItems.Add("0");

}

private static void PlayersListViewDisplayBancrupcy(Player player, ListBox listBoxEvents)

{

if (player.PlayerBancrupt == true)

{

player.ListViewItemPlayer.SubItems[1].Text = "Player Bancrupt";

listBoxEvents.Items.Add(player.PlayerName + " bancrupted");

}

}

private static void PlayersListViewAddPlayer(Player player, ListView playersListView)

{

ListViewItem listViewItem = new ListViewItem(player.PlayerName, 0);

listViewItem.SubItems.Add(player.PlayerBudget.ToString());

listViewItem.SubItems.Add(player.PlayerBalance.ToString());

//this property enables setting different colors for subitems

listViewItem.UseItemStyleForSubItems = false;

playersListView.Items.Add(listViewItem);

player.ListViewItemPlayer = listViewItem;

}

private static void PlayersListViewUpdate(List<Player> players, ListBox listBoxEvents, Dictionary<string, Company> companies, ListView playersListView)

{

if (players.Count <= 0)

return;

//updates the columns of each company

PlayersListViewColumnsUpdate(companies, playersListView, players);

foreach (Player player in players)

{

if(player.ListViewItemPlayer == null || !playersListView.Items.Contains(player.ListViewItemPlayer))

{

PlayersListViewAddPlayer(player, playersListView);

}

//adds the subitems for new columns, for each player

PlayersListViewAddSubItems(player, playersListView);

ListViewItem listViewItem = player.ListViewItemPlayer;

if (player.PlayerBancrupt)

continue;

try

{

//starting indexing with 1, because 0 is a player name

listViewItem.SubItems[1].Text = player.PlayerBudget.ToString();

listViewItem.SubItems[2].Text = player.PlayerStocksValue.ToString();

listViewItem.SubItems[3].Text = player.PlayerBalance.ToString();

//updates the numbers of shares

for(int i = 3; i < playersListView.Columns.Count; i++)

{

string columnText = playersListView.Columns[i].Text;

string companySymbol = columnText.Substring(0, columnText.Length - 7);

if (player.PlayerStocks.ContainsKey(companySymbol))

{

listViewItem.SubItems[i].Text = player.PlayerStocks[companySymbol].NumberOfShares.ToString();

}

}

}

catch (IndexOutOfRangeException e)

{

throw new ArgumentOutOfRangeException("index parameter is out of range.", e);

}

}

}

private static void PlayersListViewReset(ListView listView)

{

foreach (ListViewItem listItem in listView.Items)

{

listView.Items.Remove(listItem);

}

//removes the "numbers of shares" columns

for(int i = 3; i < listView.Columns.Count; i++)

{

listView.Columns[i].Dispose();

}

}

private static readonly Random getRandom = new Random();

public static double GetRandomNumber(double min, double max)

{

Random random = new Random(Guid.NewGuid().GetHashCode());

return random.NextDouble() \* (max - min) + min;

}

//Random number generator

private void buttonExit\_Click(object sender, EventArgs e)

{

Close();

}

private void buttonShowStocks\_Click(object sender, EventArgs e)

{

panelStockValues.Visible = true;

if (!DictCompanies.ContainsKey("aapl"))

DictCompanies.Add("aapl", Apple);

if(!DictCompanies.ContainsKey("msft"))

DictCompanies.Add("msft", Microsoft);

CompaniesListViewReset(companiesListView);

CompaniesListViewInitialize(DictCompanies, companiesListView);

LstPlayers.Add(Player1);

LstPlayers.Add(Player2);

PlayersListViewUpdate(LstPlayers, listBoxEvents, DictCompanies, playersListView);

}

private void buttonBack\_Click(object sender, EventArgs e)

{

panelStockValues.Visible = false;

}

private void timer1\_Tick(object sender, EventArgs e)

{

int threadId = Thread.CurrentThread.ManagedThreadId;

lock (locker)

{

Count++;

//We want to be sure, that every player has every company's stocks. Even if it's number of shares is 0. (temporary)

Player.FillPlayerStocksWithCompanies(DictCompanies, lstPlayers);

//multiplepurpose check value for daily changes cycle

int i = 0;

//Floor and Ceiling for Stock value changes

double floor = -0.05, ceiling = 0.05;

double change;

//Bear and Bull Markets

BullAndBearCheck = BullAndBear.eventCheckBullAndBear(GetRandomNumber(0, 1), Math.Round(GetRandomNumber(5, 15)));

switch (BullAndBearCheck)

{

case -1:

floor = -0.08; ceiling = 0;

listBoxEvents.Items.Add("Day " + Count.ToString() + "(Bear Market)");

break;

case 0:

floor = -0.05; ceiling = 0.05;

listBoxEvents.Items.Add("Day " + Count.ToString());

break;

case 1:

floor = 0; ceiling = 0.08;

listBoxEvents.Items.Add("Day " + Count.ToString() + "(Bull Market)");

break;

}

//Crash check + Floor and Ceiling changes if crash happens

if (BullAndBearCheck != 1)

if (Crash.eventCheckCrash(GetRandomNumber(0, 1)) == 1)

{

floor = -0.2;

ceiling = -0.05;

listBoxEvents.Items.Add(Crash.EventMessageBoxText + Count.ToString());

foreach (KeyValuePair<string, Company> company in DictCompanies) company.Value.Stability -= 0.01;

}

foreach (KeyValuePair<string, Company> item in DictCompanies)

{

Company company = item.Value;

if (company.CompanyDelisted != true)

{

EventDelisting.eventCheckDelisting(company, Count, GetRandomNumber(0, 1));

if (company.Stability == 0)

{

foreach (Player player in LstPlayers)

{

player.PlayerStocksValue -= (company.StockValue \* player.PlayerStocks[company.CompanySymbol].NumberOfShares);

player.PlayerStocks[company.CompanySymbol].NumberOfShares = 0;

}

company.CompanyDelisted = true;

company.StockValue = 0;

company.StockValue = 0;

CompaniesListViewDelisting(company, listBoxEvents);

}

}

if (company.CompanyDelisted != true)

{

//stock values + labels + value of stock owned by players changes

change = GetRandomNumber(floor, ceiling);

company.DailyChange(change);

foreach (Player player in LstPlayers)

{

try

{

player.PlayerStocksValue += player.PlayerStocks[item.Key].NumberOfShares \* company.DailyChangeUsd;

}

catch (KeyNotFoundException ex)

{

throw new ArgumentException("Key Not Found", "company.Key", ex);

}

}

}

}

CompaniesListViewUpdate(DictCompanies, listBoxEvents);

i = 0;

foreach (Player player in LstPlayers)

{

if (player.PlayerBancrupt != true)

{

//player buying and selling stocks + displaying it

player.PlayerBuyAndSell(DictCompanies, Count, Convert.ToInt32(GetRandomNumber(1, 15)));

if (player.PlayerBancrupt)

PlayersListViewDisplayBancrupcy(player, listBoxEvents);

PlayersListViewUpdate(LstPlayers, listBoxEvents, DictCompanies, playersListView);

}

i++;

}

i = 0;

labeldayn.Text = Count.ToString();

//simulation ending conditions

foreach (KeyValuePair<string, Company> company in DictCompanies)

if (company.Value.CompanyDelisted != true) i = 1;

if (i == 0)

{

timer1.Stop();

listBoxEvents.Items.Add("Simulation ended by delisting of all stocks");

}

i = 0;

foreach (Player player in LstPlayers) if (player.PlayerName != "Player bancrupt") i = 1;

if (i == 0)

{

timer1.Stop();

listBoxEvents.Items.Add("Simulation ended by bancrupcy of all players");

}

if (Count == 100)

{

timer1.Stop();

listBoxEvents.Items.Add("Simulation ended by reaching 100 days");

}

}

}

private void buttonStartTimer\_Click(object sender, EventArgs e)

{

timer1.Start();

PlayerRand broker1 = new PlayerRand("Broker1", 10000);

PlayerStrat broker2 = new PlayerStrat("Broker2", 10000);

Thread playerThread1 = new Thread(() => {

broker1.PlayerBuyAndSell(new Dictionary<string, Company>(), 0, Convert.ToInt32(GetRandomNumber(1, 15)));

});

playerThread1.IsBackground = true;

Thread playerThread2 = new Thread(() => {

broker2.PlayerBuyAndSell(new Dictionary<string, Company>(), 0, Convert.ToInt32(GetRandomNumber(1, 15)));

});

playerThread2.IsBackground = true;

}

private void buttonStopTimer\_Click(object sender, EventArgs e)

{

timer1.Stop();

}

private void buttonReset\_Click(object sender, EventArgs e)

{

foreach (Player player in LstPlayers)

player.PlayerReset();

PlayersListViewUpdate(LstPlayers, listBoxEvents, DictCompanies, playersListView);

foreach (KeyValuePair<string, Company> company in DictCompanies)

{

company.Value.CompanyReset();

}

CompaniesListViewReset(companiesListView);

Count = 0;

timer1.Stop();

labeldayn.Text = "0";

listBoxEvents.Items.Clear();

BullAndBear.BullAndBearDaysLeft = 0;

}

}

}