***Лабараторна робота №2***

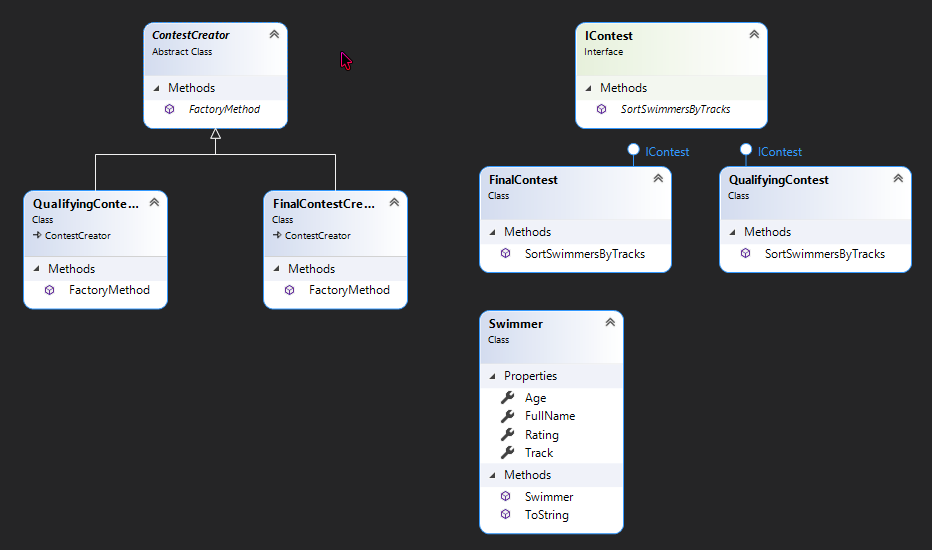
***Варіант №3***

**Завдання №1**

Застосувати паттерн Фабричний метод для створення програми, яка організовує розташування плавців на доріжках в басейні при проведенні змагань. Нехай існує клас плавця, який містить описання ПІБ, вік, місце в рейтингу та доріжка, по якій буде пливти плавець після жеребкування. Розташування плавців на доріжках можуть бути 2 видів: послідовний від найслабшого – 1 доріжка до найшвидшого – остання доріжка та симетричний – 3 найсильніші плавці розташовують в центральних доріжках, найслабші розташовують симетрично до них зліва-направо. Також існує клас, який задає інтерфейс для класів змагань. Змагання можуть бути відбірковими та фінальними, в рамках паттерну Фабричний метод ці класи виконують ролі конкретних творців. У відбіркових змаганнях завжди використовується послідовне розташування, а у фінальних – симетричний. Створити Windows-додаток, де в залежності від виду змагань видається список із розташуванням плавців.

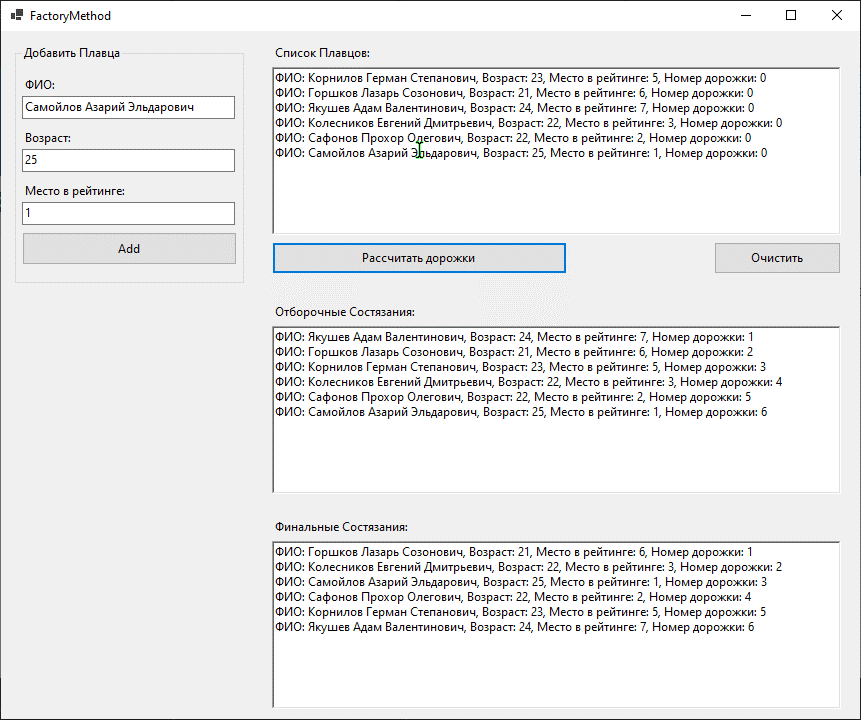
***Результати виконання завдання 1:***

*Діаграма класів додатку (рис.1):*



**Рис. 1 Діаграма класів додатку**

*Результати роботи додатку (рис.2)*



**Рис. 2 Результат виконаня додатку**

*Лістинг програмного коду:*

*\*\* MainForm.cs \*\**

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace Lab\_2\_1

{

public partial class MainForm : Form

{

List<Swimmer> \_swimmers;

public MainForm()

{

InitializeComponent();

\_swimmers = new List<Swimmer>();

}

private string swimmersListToString(List<Swimmer> swimmers)

{

string result = "";

foreach (var swimmer in swimmers)

{

result += swimmer + "\r\n";

}

return result;

}

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

{

string fullName = fullNameTextBox.Text;

int age;

int rating;

if (fullName == "" || fullName == "Введите ФИО")

{

fullNameTextBox.Text = "Введите ФИО";

return;

}

try

{

age = Convert.ToInt32(ageTextBox.Text);

}

catch (Exception)

{

ageTextBox.Text = "Введите число";

return;

}

try

{

rating = Convert.ToInt32(ratingTextBox.Text);

}

catch (Exception)

{

ratingTextBox.Text = "Введите число";

return;

}

\_swimmers.Add(new Swimmer(fullName, age, rating));

swimmersListRichTextBox.Text = swimmersListToString(\_swimmers);

}

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

{

if (\_swimmers.Count <= 0)

{

MessageBox.Show("Для начала добавтьте плавцов через форму с лева.", "Ошибка");

}

else

{

ContestCreator qualifyingContestCreator = new QualifyingContestCreator();

ContestCreator finalContestCreator = new FinalContestCreator();

IContest qualifyingContest = qualifyingContestCreator.FactoryMethod();

IContest finalContest = finalContestCreator.FactoryMethod();

QualifyingContestTextBox.Enabled = true;

FinalContestTextBox.Enabled = true;

QualifyingContestTextBox.Text = swimmersListToString(qualifyingContest.SortSwimmersByTracks(\_swimmers));

FinalContestTextBox.Text = swimmersListToString(finalContest.SortSwimmersByTracks(\_swimmers));

}

}

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

{

\_swimmers.Clear();

swimmersListRichTextBox.Text = "";

QualifyingContestTextBox.Text = "";

FinalContestTextBox.Text = "";

QualifyingContestTextBox.Enabled = false;

FinalContestTextBox.Enabled = false;

MessageBox.Show("Список плавцов был очищен.", "Очищенно");

}

}

}

*\*\* ContestFactory.cs \*\**

using System;

using System.Collections.Generic;

namespace Lab\_2\_1

{

abstract class ContestCreator

{

public abstract IContest FactoryMethod();

}

class QualifyingContestCreator : ContestCreator

{

public override IContest FactoryMethod()

{

return new QualifyingContest();

}

}

class FinalContestCreator : ContestCreator

{

public override IContest FactoryMethod()

{

return new FinalContest();

}

}

public interface IContest

{

List<Swimmer> SortSwimmersByTracks(List<Swimmer> swimmers);

}

class QualifyingContest : IContest

{

public List<Swimmer> SortSwimmersByTracks(List<Swimmer> swimmers)

{

List<Swimmer> newSwimmers = new List<Swimmer>();

swimmers.ForEach((item)=>

{

newSwimmers.Add(new Swimmer(item.FullName, item.Age, item.Rating));

});

newSwimmers.Sort((xSwimmer, ySwimmer) =>

{

if (xSwimmer.Rating < ySwimmer.Rating) return 1;

if (xSwimmer.Rating > ySwimmer.Rating) return -1;

return xSwimmer.Rating.CompareTo(ySwimmer.Rating);

});

int track = 1;

foreach (Swimmer swimmer in newSwimmers)

{

swimmer.Track = track;

track++;

}

return newSwimmers;

}

}

class FinalContest : IContest

{

public List<Swimmer> SortSwimmersByTracks(List<Swimmer> swimmers)

{

List<Swimmer> newSwimmers = new List<Swimmer>();

List<Swimmer> tempSwimmers = new List<Swimmer>();

swimmers.ForEach((item)=>

{

newSwimmers.Add(new Swimmer(item.FullName, item.Age, item.Rating));

});

newSwimmers.Sort((xSwimmer, ySwimmer) =>

{

if (xSwimmer.Rating > ySwimmer.Rating) return 1;

if (xSwimmer.Rating < ySwimmer.Rating) return -1;

return xSwimmer.Rating.CompareTo(ySwimmer.Rating);

});

bool flag = false;

foreach (Swimmer swimmer in newSwimmers)

{

if (!flag)

{

tempSwimmers.Insert(0, swimmer);

flag = true;

}

else

{

tempSwimmers.Add(swimmer);

flag = false;

}

}

int track = 1;

foreach (Swimmer swimmer in tempSwimmers)

{

swimmer.Track = track;

track++;

}

return tempSwimmers;

}

}

}

*Лістинг програмного коду:*

using System;

using System.Windows.Forms;

namespace Lab\_1

{

abstract class Weapon

{

protected string Name { get; set; }

protected string OwnerName { get; set; }

protected double Damage { get; set; }

protected Weapon(string name, string ownerName, double damage)

{

Name = name;

OwnerName = ownerName;

Damage = damage;

}

public virtual string Info()

{

return $"Название: {Name}, Владелец: {OwnerName}, Урон: {Damage}";

}

public abstract string TakeDamage(string To);

}

class Knife : Weapon

{

private readonly double \_length;

public Knife(string name, string ownerName, double damage, double length) : base(name, ownerName, damage)

{

\_length = length;

}

public override string Info()

{

return $"Название: {Name}, Владелец: {OwnerName}, Урон: {Damage}, Длина: {\_length}";

}

public override string TakeDamage(string to)

{

return $"{OwnerName} ударил {Name} {to} и нанес(ла) {Damage} урона.";

}

}

class ShotGun : Weapon

{

private readonly double \_length;

private int \_bulletsCount;

public ShotGun(string name, string ownerName, double damage, double length, int bulletsCount) : base(name,

ownerName, damage)

{

\_length = length;

\_bulletsCount = bulletsCount;

}

public override string Info()

{

return $"Название: {Name}, Владелец: {OwnerName}, Урон: {Damage}, Длина: {\_length}";

}

public override string TakeDamage(string to)

{

if (\_bulletsCount > 0)

{

\_bulletsCount--;

return $"{OwnerName} выстрелил из {Name} в {to} и нанес(ла) {Damage} урона.";

}

return $"{OwnerName} не смог(ла) выстрелить из {Name} в {to}. Нет патронов!";

}

public string addBullets(int count)

{

\_bulletsCount += count;

return $"{OwnerName} зарядил в {Name} {count} патронов. Текущее количество: {\_bulletsCount}";

}

}

public partial class MainForm : Form

{

private Weapon \_weapon;

private string damageTextBoxLastText = "0";

private string lengthTextBoxLastText = "0";

private string bulletsCountTextBoxLastText = "15";

public MainForm()

{

InitializeComponent();

}

private void MainForm\_Load(object sender, EventArgs e)

{

weaponTypeBox.Items.Add("Нож");

weaponTypeBox.Items.Add("Дробовик");

weaponTypeBox.SelectedIndex = 0;

damageTextBox.Text = damageTextBoxLastText;

lengthTextBox.Text = lengthTextBoxLastText;

bulletsCountTextBox.Text = bulletsCountTextBoxLastText;

}

private void PrintToLogs(string msg)

{

logsTextBox.Text += msg;

logsTextBox.Text += '\n';

}

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

{

if (weaponTypeBox.SelectedIndex == 1)

{

ShotGun \_\_weapon = new ShotGun(

weaponNameTextBox.Text,

nameTextBox.Text,

double.Parse(damageTextBox.Text),

double.Parse(lengthTextBox.Text),

int.Parse(bulletsCountTextBox.Text));

PrintToLogs(\_\_weapon.addBullets(25));

}

}

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

{

#region Checks

if (nameTextBox.Text.Length <= 0)

{

MessageBox.Show("Вы не ввели ваше имя.");

return;

}

if (enemyNameTextBox.Text.Length <= 0)

{

MessageBox.Show("Вы не ввели имя противника.");

return;

}

if (weaponNameTextBox.Text.Length <= 0)

{

MessageBox.Show("Вы не ввели название оружия.");

return;

}

#endregion

switch (weaponTypeBox.SelectedIndex)

{

case 0:

\_weapon = new Knife(

weaponNameTextBox.Text,

nameTextBox.Text,

double.Parse(damageTextBox.Text),

double.Parse(lengthTextBox.Text));

break;

case 1:

\_weapon = new ShotGun(

weaponNameTextBox.Text,

nameTextBox.Text,

double.Parse(damageTextBox.Text),

double.Parse(lengthTextBox.Text),

int.Parse(bulletsCountTextBox.Text));

break;

}

PrintToLogs(\_weapon.TakeDamage(enemyNameTextBox.Text));

}

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

{

logsTextBox.Text = "";

}

private void weaponTypeBox\_SelectedIndexChanged(object sender, EventArgs e)

{

switch (weaponTypeBox.SelectedIndex)

{

case 0:

label6.Hide();

bulletsCountTextBox.Hide();

reloadWeaponBtn.Hide();

break;

case 1:

label6.Show();

bulletsCountTextBox.Show();

reloadWeaponBtn.Show();

break;

}

}

private void danageTextBox\_TextChanged(object sender, EventArgs e)

{

if (double.TryParse(damageTextBox.Text, out \_))

{

damageTextBoxLastText = damageTextBox.Text;

}

else

{

damageTextBox.Text = damageTextBoxLastText;

}

}

private void lenghtTextBox\_TextChanged(object sender, EventArgs e)

{

if (double.TryParse(lengthTextBox.Text, out \_))

{

lengthTextBoxLastText = lengthTextBox.Text;

}

else

{

lengthTextBox.Text = lengthTextBoxLastText;

}

}

private void bulletsCountTextBox\_TextChanged(object sender, EventArgs e)

{

if (int.TryParse(bulletsCountTextBox.Text, out \_))

{

bulletsCountTextBoxLastText = bulletsCountTextBox.Text;

}

else

{

bulletsCountTextBox.Text = bulletsCountTextBoxLastText;

}

}

private void weaponNameTextBox\_TextChanged(object sender, EventArgs e)

{

if (weaponNameTextBox.Text.Length > 0)

{

damageTextBox.Enabled = true;

lengthTextBox.Enabled = true;

bulletsCountTextBox.Enabled = true;

takeDamageBtn.Enabled = true;

reloadWeaponBtn.Enabled = true;

}

else

{

damageTextBox.Enabled = false;

lengthTextBox.Enabled = false;

bulletsCountTextBox.Enabled = false;

takeDamageBtn.Enabled = false;

reloadWeaponBtn.Enabled = false;

}

}

}

}

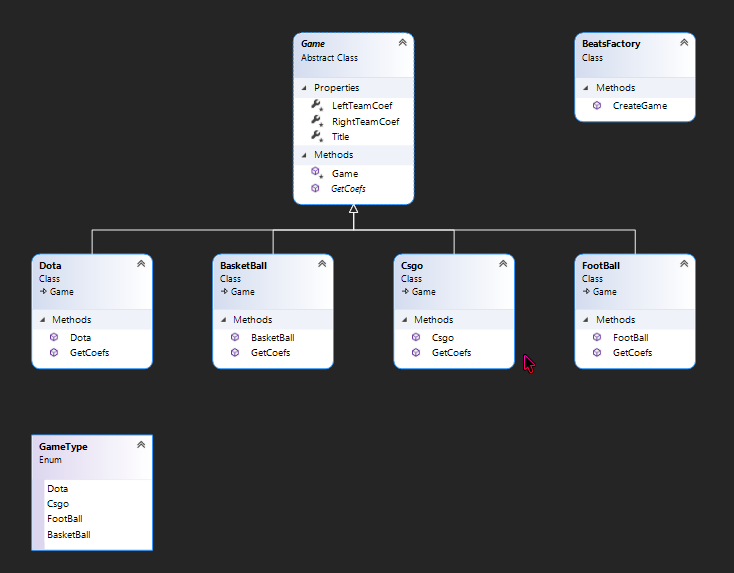
**Завдання №2**

Застосувати патерн Фабричний метод для заданої предметної області.

(3. Букмекерська контора)

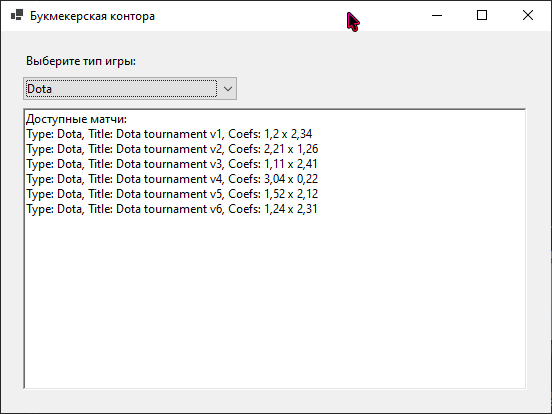
***Результати виконання завдання 2:***

*Діаграма класів додатку (рис.3):*

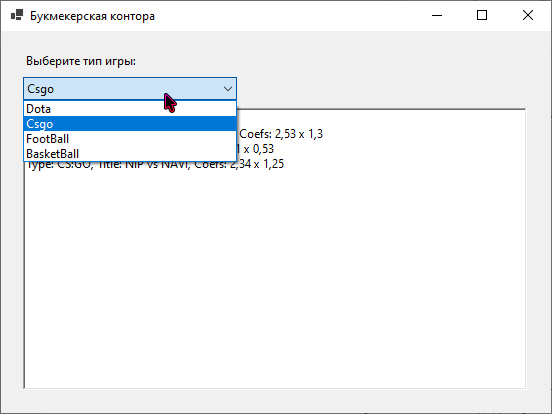


**Рис. 3 Діаграма класів додатку**

*Результати роботи додатку (рис.4, рис.5)*



**Рис. 4 Результат виконаня додатку**



**Рис. 5 Результат виконаня додатку**

*Лістинг програмного коду:*

*\*\* MainForm.cs \*\**

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace Lab\_2\_2

{

public partial class MainForm : Form

{

private BeatsFactory \_beatsFactory = new BeatsFactory();

private List<Game> \_dotaGames;

private List<Game> \_csgoGames;

private List<Game> \_footBallGames;

private List<Game> \_basketBallGames;

public MainForm()

{

InitializeComponent();

#region Create Games

\_dotaGames = new List<Game>

{

\_beatsFactory.CreateGame(GameType.Dota, "Dota tournament v1", 1.2, 2.34),

\_beatsFactory.CreateGame(GameType.Dota, "Dota tournament v2", 2.21, 1.26),

\_beatsFactory.CreateGame(GameType.Dota, "Dota tournament v3", 1.11, 2.41),

\_beatsFactory.CreateGame(GameType.Dota, "Dota tournament v4", 3.04, 0.22),

\_beatsFactory.CreateGame(GameType.Dota, "Dota tournament v5", 1.52, 2.12),

\_beatsFactory.CreateGame(GameType.Dota, "Dota tournament v6", 1.24, 2.31)

};

\_csgoGames = new List<Game>

{

\_beatsFactory.CreateGame(GameType.Csgo, "NAVI vs The Alliance", 2.53, 1.30),

\_beatsFactory.CreateGame(GameType.Csgo, "OG vs NiP", 3.11, 0.53),

\_beatsFactory.CreateGame(GameType.Csgo, "NiP vs NAVI", 2.34, 1.25)

};

\_footBallGames = new List<Game>

{

\_beatsFactory.CreateGame(GameType.FootBall, "Сербия vs Шотландия", 1.88, 4.80),

\_beatsFactory.CreateGame(GameType.FootBall, "Венгрия vs Исландия", 2.21, 4.20),

\_beatsFactory.CreateGame(GameType.FootBall, "Уэльс vs США", 2.70, 2.75)

};

\_basketBallGames = new List<Game>

{

\_beatsFactory.CreateGame(GameType.FootBall, "Гран Канария vs Тренто", 1.92, 1.89),

\_beatsFactory.CreateGame(GameType.FootBall, "Нижний Новгород vs Парма Пермь", 1.87, 1.91),

};

#endregion

gameTypeComboBox.Items.Add(GameType.Dota);

gameTypeComboBox.Items.Add(GameType.Csgo);

gameTypeComboBox.Items.Add(GameType.FootBall);

gameTypeComboBox.Items.Add(GameType.BasketBall);

gameTypeComboBox.SelectedIndex = 0;

}

private void gameTypeComboBox\_SelectedIndexChanged(object sender, EventArgs e)

{

outputTextBox.Text = "Доступные матчи:\r\n";

List<Game> selectedGames;

switch (gameTypeComboBox.SelectedIndex)

{

case 0: selectedGames = \_dotaGames; break;

case 1: selectedGames = \_csgoGames; break;

case 2: selectedGames = \_footBallGames; break;

case 3: selectedGames = \_basketBallGames; break;

default: selectedGames = new List<Game>(); break;

}

if (selectedGames.Count <= 0)

{

outputTextBox.Text = "В этом типе пока нет игр.";

}

foreach (var game in selectedGames)

{

outputTextBox.Text += $"{game.GetCoefs()}\r\n";

}

}

}

}

*\*\* BeatsFactory.cs \*\**

using System;

namespace Lab\_2\_2

{

public class BeatsFactory

{

public Game CreateGame(GameType gameType, string title, double leftTeamCoef, double rightTeamCoef)

{

Game game;

switch (gameType)

{

case GameType.Dota:

game = new Dota(title, leftTeamCoef, rightTeamCoef);

break;

case GameType.Csgo:

game = new Csgo(title, leftTeamCoef, rightTeamCoef);

break;

case GameType.FootBall:

game = new FootBall(title, leftTeamCoef, rightTeamCoef);

break;

case GameType.BasketBall:

game = new BasketBall(title, leftTeamCoef, rightTeamCoef);

break;

default:

throw new ArgumentException("An invalid Game Type: " + gameType);

}

return game;

}

}

public abstract class Game

{

protected string Title { get; }

protected double LeftTeamCoef { get; }

protected double RightTeamCoef { get; }

protected Game(string title, double leftTeamCoef, double rightTeamCoef)

{

Title = title;

LeftTeamCoef = leftTeamCoef;

RightTeamCoef = rightTeamCoef;

}

public abstract string GetCoefs();

}

public enum GameType

{

Dota,

Csgo,

FootBall,

BasketBall

}

public class Dota : Game

{

public Dota(string title, double leftTeamCoef, double rightTeamCoef)

: base(title, leftTeamCoef, rightTeamCoef) { }

public override string GetCoefs()

{

return $"Type: Dota, Title: {Title}, Coefs: {LeftTeamCoef} x {RightTeamCoef}";

}

}

public class Csgo : Game

{

public Csgo(string title, double leftTeamCoef, double rightTeamCoef)

: base(title, leftTeamCoef, rightTeamCoef) { }

public override string GetCoefs()

{

return $"Type: CS:GO, Title: {Title}, Coefs: {LeftTeamCoef} x {RightTeamCoef}";

}

}

public class FootBall : Game

{

public FootBall(string title, double leftTeamCoef, double rightTeamCoef)

: base(title, leftTeamCoef, rightTeamCoef) { }

public override string GetCoefs()

{

return $"Type: FootBall, Title: {Title}, Coefs: {LeftTeamCoef} x {RightTeamCoef}";

}

}

public class BasketBall : Game

{

public BasketBall(string title, double leftTeamCoef, double rightTeamCoef)

: base(title, leftTeamCoef, rightTeamCoef) { }

public override string GetCoefs()

{

return $"Type: BasketBall, Title: {Title}, Coefs: {LeftTeamCoef} x {RightTeamCoef}";

}

}

}