***Wojskowa Akademia Techniczna***

***im. Jarosława Dąbrowskiego***



**Wydział Cybernetyki, kierunek informatyka - inżynieria systemów**

Laboratorium z przedmiotu:

Metody programowania .NET

Sprawozdanie z projektu:

**Serwis aukcyjny**

Prowadzący:

dr inż. Krzysztof Mierzejewski

Wykonali:

**Radosław Relidzyński**

**Bartłomiej Pstrągowski**

**Grupa:** WCY20IJ1S1

**Termin oddania ćwiczenia**: 31.01.2023 r.

Spis treści

[A. Treść zadania 3](#_Toc126096413)

[Scorecard 3](#_Toc126096414)

[Wymagania do sprawozdania 4](#_Toc126096415)

[B. Opis zaimplementowanego rozwiązania 4](#_Toc126096416)

[Serwer 4](#_Toc126096417)

[Klient 4](#_Toc126096418)

[Aukcje 5](#_Toc126096419)

[Cykl życia aukcji 5](#_Toc126096420)

[Diagram przypadków użycia 6](#_Toc126096421)

[Diagram UML **Błąd! Nie zdefiniowano zakładki.**](#_Toc126096422)

[C. Konfiguracja środowiska 6](#_Toc126096423)

[D. Demonstracja działania programu](#_Toc126096424)

[E. Zastosowane mechanizmy z punktów Scorecard](#_Toc126096425)

[1. Dwustronna komunikacja międzyprocesowa.](#_Toc126096426)

[2. Kontrakty](#_Toc126096427)

[2.1. Klasy abstrakcyjne](#_Toc126096428)

[2.2. Interfejsy](#_Toc126096429)

[3. Polimorfizm](#_Toc126096430)

[3.1. Uogólnianie typów](#_Toc126096431)

[3.2. Przesłanianie metod](#_Toc126096432)

[4. Typy wyliczeniowe](#_Toc126096433)

[5. Typy generyczne](#_Toc126096434)

[6. Atrybuty](#_Toc126096435)

[7. Metody anonimowe, wyrażenia Lambda](#_Toc126096436)

[8. Language Integrated Query (LINQ)](#_Toc126096437)

[9. Metody async, synchronizacja await lub Task API](#_Toc126096438)

[10. Wykorzystanie wyrażeń regularnych](#_Toc126096439)

[11. Implementacja i prawidłowe wykorzystanie interfejsu IDisposable](#_Toc126096440)

[12. Extension method dla typu Base Class Library](#_Toc126096441)

[F. Prezentacja w pełni automatycznej komunikacji międzyprocesowej](#_Toc126096442)

[G. Wnioski](#_Toc126096443)

# Treść zadania

Wymagania na projekt

1. Przekazać należy archiwum z kompletną solucją dla JetBrains Rider zawierającą pełne kody źródłowe obu projektów wraz z (opcjonalnie) zależnościami pakietów NuGet oraz bibliotek zewnętrznych umożliwiając ocenę tak statyczną (code review) jak i dynamiczną (kompilacja oraz uruchomienie).
2. Target framework: net6.0; konsola / terminal jako interfejs użytkownika (input / output); nie należy zakładać wykorzystania konkretnego systemu operacyjnego / platformy do uruchomienia rozwiązania (np. MacOS).
3. W ramach opisanej w sprawozdaniu demonstracyjnej ścieżki uruchomienia rozwiązania musi nastąpić przynajmniej jednokrotna, obustronna (duplex) komunikacja międzyprocesowa pomiędzy projektami (tj. z A→B i z B→A). Jeżeli komunikacja wyzwalana jest manualnie, tj. odbiorca komunikatu wymaga ręcznego działania w celu pobraniu danych, maksymalna możliwa ocena za takie rozwiązanie wynosi 4,5.
4. Dwie płaszczyzny oceny:

* projekt solucji, sprawozdanie i działanie rozwiązania jako całość;
* kod źródłowy obu projektów z wykorzystaniem Scorecard oceniany osobno.

### Scorecard

**Ocena db.**

Dwustronna komunikacja międzyprocesowa

**Programowanie obiektowe**

Kontrakty: klasy abstrakcyjne, interfejsy

Polimorfizm: uogólnianie typów, przesłanianie metod

Typy wyliczeniowe

Zaprojektowanie typów generycznych, kowariancja i kontrawariancja

**Metadane**

Atrybuty: własne lub wykorzystanie istniejących

**Programowanie funkcyjne**

Delegaty: metody anonimowe, wyrażenia lambda

Kolekcje danych i Language Integrated Query

**Programowanie asynchroniczne**

Metody async, synchronizacja await lub Task API

**+ 1/2 do oceny**

**Przynajmniej 3 punkty z poniższej listy:**

1. Wykorzystanie wyrażeń regularnych

2. Implementacja i prawidłowe wykorzystanie interfejsu IDisposable

3. Extension metods dla typów z Base Class Library

4. Użycie CancellationToken

5. Wykorzystanie synchronization primitives

**+ 1/2 do oceny**

Komunikacja międzyprocesowa jest w pełni automatyczna, tj. nie wymaga ręczne

wyzwalania pobierania komunikatów po stronie odbiorcy.

**- 1/2 do oceny**

Nie uwalnianie zasobów z interfejsem IDisposable, jeżeli jest on dostępny w

wykorzystywanych obiektach.

### Wymagania do sprawozdania

* + 1. Ogólny opis zaimplementowanego rozwiązania / use case.
    2. (O ile występują) specyficzne wymagania odnośnie konfiguracji środowiska w celu uruchomienia i dynamicznej oceny rozwiązania.
    3. Opis demonstracyjnej ścieżki uruchomienia rozwiązania.
    4. Dla każdego z projektów osobno: wskazanie w kodzie źródłowym wraz z krótkim uzasadnieniem dla zastosowania mechanizmu każdego pokrytego ze Scorecard punktu.

# Opis zaimplementowanego rozwiązania

Projekt jest serwisem aukcyjnym, w ramach którego klienci mogą tworzyć aukcje oraz je licytować. Poniżej opisuję kluczowe elementy projektu przedstawiające, co dany element posiada oraz jakie ma możliwości:

### Serwer

* Przechowuje informacje o aukcjach i klientach
* Obsługuje działania klientów
* Pilnuje czasu każdej aukcji (jeśli minie czas, kończy ją)

### Klient

* Posiada id, nazwę oraz budżet
* Ma możliwość tworzenia aukcji
* Klienci dzielą się na 2 grupy: Licytator oraz Licytator rangi VIP (potrzebna jest ranga VIP, aby móc tworzyć aukcje powyżej wartości 1000)
* Ma możliwość przeglądania wszystkich aktualnych aukcji
* Dla każdej aukcji może licytować w dowolnej kwocie nie przekraczającej jego własnego budżetu

### Aukcje

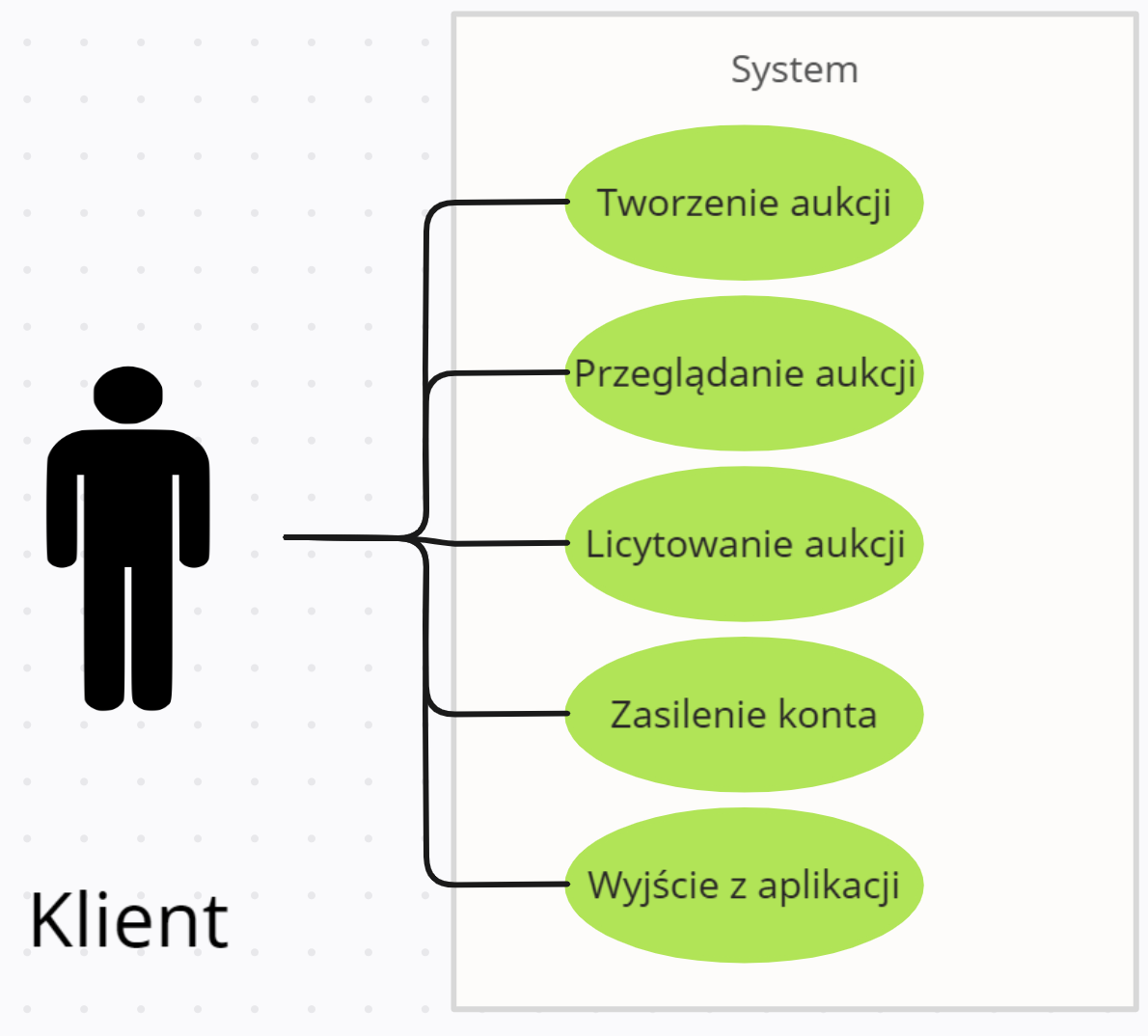
* Każda ma id, nazwę przedmiotu, id właściciela, id aktualnie wygrywającego klienta (początkowo puste), czas zakończenia aukcji

Działanie programu obraca się wokół aukcji, gdzie każda z nich ma określony cykl życia, opisany poniżej:

### Cykl życia aukcji

1. Klient wysyła zapytanie o utworzenie aukcji.
2. Jeśli dane są poprawne, powstaje aukcja.
3. Czas zakończenia aukcji ustalany jest na podstawie godziny utworzenia aukcji oraz czasu trwania aukcji podanego przez klienta.
4. Klienci licytują:
   * Nie można licytować kwotą nie większą niż jest podana
   * Pierwsza podana stawka ustawia od razu pierwszego zwycięzcę
   * Każde podbicie stawki powoduje, że klientowi pobierana jest odpowiednia kwota
   * Jeśli ktoś przebił stawkę klienta, zwracane mu są pieniądze.
5. W tle program na bieżąco monitoruje, czy czas zakończenia aukcji nie minął.
6. Jeśli minął czas aukcji, zostaje ona zamknięta, a właściciel aukcji otrzymuje należne mu pieniądze.
7. Aukcja znika i już jest niedostępna.

### Diagram przypadków użycia



# Konfiguracja środowiska

* + 1. W prawym górnym rogu wybrać opcję „AuctionHouseSerwer”.



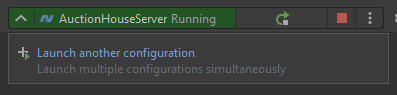


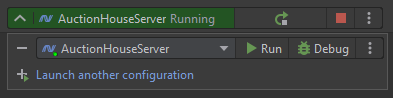
* + 1. Kliknąć „Run” (uruchomić serwer).

Obraz zawierający tekst

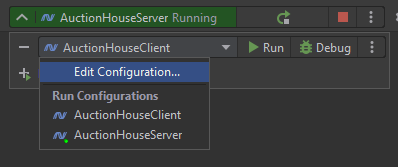
Opis wygenerowany automatycznie

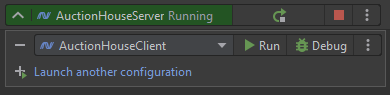
* + 1. W prawym górnym rogu rozwinąć opcje dodatkowe i kliknąć „Launch another configuration”.



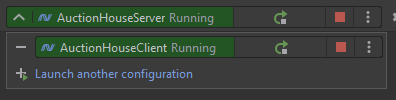


* + 1. W drugiej opcji wybrać „AuctionHouseClient”.





* + 1. Kliknąć „Run” (uruchomić klienta).



Obraz zawierający tekst

Opis wygenerowany automatycznie

* + 1. Wprowadzić nazwę klienta.

Obraz zawierający tekst

Opis wygenerowany automatycznie

* + 1. W tym momencie klient jest już podpięty do serwera, może wykonywać na nim opcje

Klientów można dodać kilka, dla każdego nowego klienta należy wykonać kroki 5-7.

# Demonstracja działania programu

Obraz zawierający tekst

Opis wygenerowany automatycznie

Serwer działa, przykładowo zostały utworzone 2 instancje klientów z czego 2 już podłączyła się do serwera.

Obraz zawierający tekst, zrzut ekranu, monitor, czarny

Opis wygenerowany automatycznie

Jeden klient próbował wyświetlić listę aukcji natomiast drugi zaczął tworzyć nową aukcję.

Obraz zawierający tekst, zrzut ekranu, monitor

Opis wygenerowany automatycznie

Klient znowu spróbował wyświetlić aukcje, tym razem istnieje już stworzona aukcja.

Obraz zawierający tekst, zrzut ekranu, monitor, biurko

Opis wygenerowany automatycznie

Przelicytował aukcję po czym rozłącza się

Obraz zawierający tekst, monitor, zrzut ekranu, wewnątrz

Opis wygenerowany automatycznie

Twórca aukcji sprawdza stan aukcji ale okazuje się, że już się ona zakończyła

Obraz zawierający tekst, monitor, zrzut ekranu, komputer

Opis wygenerowany automatycznie

Przychodzi 2 nowych klientów. Jeden z nich tworzy aukcje

Obraz zawierający tekst, zrzut ekranu, monitor, sprzęt elektroniczny

Opis wygenerowany automatycznie

Klient 1 dał pierwszą ofertę w wysokości 2000

Obraz zawierający tekst

Opis wygenerowany automatycznie

Jeden z klientów próbuje przebić ofertę innego, ale nie ma tyle pieniędzy na koncie aukcyjnym. Próbuje doładować konto.

Obraz zawierający tekst

Opis wygenerowany automatycznie

I przebija ofertę poprzenika

Obraz zawierający tekst

Opis wygenerowany automatycznie

Niestety oferta wygasła, bo minął jej czas trwania.

# Zastosowane mechanizmy z punktów Scorecard

### Dwustronna komunikacja międzyprocesowa.

Uzyskania poprzez wprowadzenie obiektów „Named Pipes”. Serwer tworzy pipe, który będzie wykorzystywany do przyłączania kolejnych klientów. Klient po uruchomieniu przyłącza się do serwera i przez stworzonego przez Serwer pipe wysyła mu infomację o swojej obecności. Wtedy serwer tworzy nowy pipe do komunikacji z danym klientem. Od tego momentu będą się komunikować ze sobą za pomocą tego prywatnego łącza Named Pipe. Komunikacja za jego pomocą jest sekwencyjna – w momencie kiedy Klient wyrazi chęć wykonania jakiejś akcji wysyła on informację do serwera który automatycznie odbiera informacje wykonuje zlecenie oraz wysyła odpowiedź, którą klient wypisuje na ekran. Dane wysyłane przez pipe są serializowane do JSON stringów.

[PREZENTACJA]

Pierwsze podłączenie się do serwera w celu uzyskania trwałego połączenia

var pipe = new NamedPipeClientStream(".", "demo2pipe", PipeDirection.**InOut**);  
  
Console.WriteLine("Waiting for connection");  
while (!pipe.IsConnected)  
{  
 try  
 {  
 pipe.Connect();  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e);  
 throw;  
 }  
 Thread.Sleep(100);  
}  
Console.WriteLine("Connected to the sever");

Klient podłącza się do swojej prywatnej linii:

\_privateCommunicationPipe = new NamedPipeClientStream(".", \_name, PipeDirection.**InOut**);  
  
Console.WriteLine("Connecting to private line");  
while (!\_privateCommunicationPipe.IsConnected)  
{  
 try  
 {  
 \_privateCommunicationPipe.Connect();  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e);  
 throw;  
 }  
 Thread.Sleep(100);  
}  
Console.WriteLine("Connected to the private communication line");

Klient wykorzystuje obiekt typu StreamString w celu przesyłania danych przez pipe’a do serwera:

var pipeWr = new StreamString(\_privateCommunicationPipe);  
  
while (\_loop)  
{  
 Menu.DrawMenu(\_name);  
 string ask = Menu.Select();  
  
 pipeWr.WriteString(ask);  
 \_privateCommunicationPipe.Flush();

Następnie czeka i pobiera komunikat zwrotny który jest deserializowany w celu uzyskania obiektu odpowiedzi:

var msg = pipeWr.ReadString();  
var reply = JsonSerializer.Deserialize<Response>(msg);

Serwer z kolei, czeka na podłączenie się klienta w celu stworzenia mu prywatnej linii komunikacji:

while(true)  
{  
 if (\_pipeServer.IsConnected())  
 {  
 \_pipeServer.WaitConnection();  
 msg = \_pipeServer.Read();  
   
 \_pipeServers.Add(new PrivPipe(new PipeServer(msg)));

W razie potrzeby tworzy nowe Task’i komunikacji z klientami:

pipeServer.Task = new Task(() =>  
{  
 Communicate(pipeServer.Pipe);  
});  
  
pipeServer.Task.Start();

W metodzie komunikacji oczekuje na wiadomość która jest następnie dekodowana

if (pipe.IsConnected() && !pipe.ToDelete)  
{  
 pipe.WaitConnection();  
 var msg = pipe.Read();  
 var message = new Response();  
   
 var rec = JsonSerializer.Deserialize<Base>(msg);

I automatycznie po jej przetworzeniu wysyłana jest odpowiedź

pipe.WriteIfConnected(JsonSerializer.Serialize(message));

### Kontrakty

### Klasy abstrakcyjne

W projekcie zastosowano klasę abstrakcyjną „Client”. Jej deklaracja posiada słowo kluczowe „abstract”, przez co nie ma możliwości utworzenia obiektu tej klasy. Klasy ją dziedziczące to „Auctioneer” oraz „AuctioneerVIP”, na podstawie których możemy tworzyć obiekty. Obiekty te będą posiadały atrybuty oraz metody zaimplementowane w ramach klasy nadrzędnej „Client”.

Klasa „Client”, jej deklaracja, atrybuty i konstruktor:

public abstract class Client  
{  
 protected readonly int Id;  
 private int \_money;  
 private static int \_idCounter = 1;  
 private readonly string \_name;  
  
 protected Client(int initMoney, string name)  
 {  
 Id = \_idCounter++;  
 \_money = initMoney;  
 \_name = name;  
 }

Klasa „Auctioneer”, jej deklaracja i konstruktor

public class Auctioneer : Client  
{  
  
 public Auctioneer(int initMoney, string name) : base(initMoney,name)  
 {  
 Console.WriteLine("Auctioneer with id {0} crated", Id);  
 }

Klasa „AuctioneerVIP”, jej deklaracja i konstruktor

public class AuctioneerVip : Client  
{  
 public AuctioneerVip(int initMoney,string name) : base(initMoney,name)  
 {  
 Console.WriteLine("AuctioneerVIP with id {0} crated", Id);  
 }

Tworzenie obiektów tych klas w programie

private readonly List<Client> \_clientsList;

…

\_clientsList = new List<Client>();

…

\_clientsList.Add(new AuctioneerVip(2000, msg));

…

\_clientsList.Add(new Auctioneer(2000, msg));

### Interfejsy

Projekt zawiera implementację interfejsu „IValidateName”. Zawiera ona funkcję „ValidateName” służącą do sprawdzania poprawności nazwy aukcji w momencie jej tworzenia. Stąd właśnie klasa „Auction” implementuje go i wykorzystuje w swoim konstruktorze do upewnienia się przed stworzeniem obiektu, czy nazwa jest poprawna.

Tworzenie interfejsu:

public interface IValidateName  
 {  
 bool ValidateName(string name);  
 }  
}

Deklaracja dziedziczenia interfejsu przez klasę „Auction”

public sealed class Auction : IValidateName, IDisposable

Implementacja metody z interfejsu w klasie „Auction”

public bool ValidateName(string name)  
{  
 var regex = new Regex("^[A-Z][a-z]+\\b(?:\\s+[A-Z][a-z]+\\b)\*$");  
 return regex.IsMatch(name);  
}

Użycie zaimplementowanej metody

if (!ValidateName(name.Trim()))  
{  
 Console.WriteLine("Error, wrong auction name input ({0})", name);  
 Id = -1;  
}

### Polimorfizm

### Uogólnianie typów

Klasy „Auctioneer” oraz „AuctioneerVIP” są klasami dziedziczącymi po klasie „Client”. W momencie tworzenia nowego obiektu niezależnie której z tych dwóch podklas, wstawiany jest do listy o typie „Client”. Dlatego każde wywołanie dowolnej metody odbywa się z poziomu klasy „Client”, mimo, że obiekty te nie są tworzone bezpośrednio z niej (gdyż jest to klasa abstrakcyjna).

Tworzenie listy klientów

private readonly List<Client> \_clientsList;

…

\_clientsList = new List<Client>();

Dodawanie obiektów różnych podklas do listy klientów

\_clientsList.Add(new AuctioneerVip(2000, msg));

…

\_clientsList.Add(new Auctioneer(2000, msg));

Wywoływanie metod dla listy klientów niezależnie od rodzaju podklasy

message.Message = \_clientsList.Find(c => c.GetName() == pipe.GetName()).CreateAuction(  
 received.Name, received.Value, received.Time, \_auctionList);

…

message.Message = \_clientsList.Find(c => c.GetName() == pipe.GetName())  
 .Bid(received3.auctionId, received3.bidValue, \_auctionList, \_clientsList);

…

message.Message = \_clientsList.Find(c => c.GetName() == pipe.GetName())  
 .UpdateMoney('+', received4.value);

### Przesłanianie metod

Klasa „Client” zawiera deklarację metody „CreateAuction”, natomiast nie posiada jej implementacji. Dopiero klasy dziedziczące implementują tą metodę przysłaniając ją przy pomocy deklaracji ze słowem kluczowym „override”. Stąd wywołanie tej metody spowoduje, że wykonają się instrukcje zadeklarowane w danej podklasie.

Deklaracja metody w klasie nadrzędnej

public abstract string CreateAuction(string auctionName, int initialValue, int auctionTime,  
 AuctionList<int> auctionList);

Nadpisanie metody w klasach podrzędnych

public override string CreateAuction(string auctionName, int initialValue, int auctionTime,  
 AuctionList<int> auctionList)  
{  
 if (initialValue > 1000)  
 {  
 *//Console.WriteLine("Error, Non-VIP auctioneer is not allowed to create auctions with that high initial amount, id: {0}", Id);* return  
 "Error, Non-VIP auctioneer is not allowed to create auctions with that high initial amount, id: " +  
 Id;  
 }  
 else  
 {  
 var auctionId = auctionList.CreateAuction(auctionName, initialValue, auctionTime, Id);  
 if (auctionId != -1)  
 {  
 *//Console.WriteLine("Auctioneer {0} successfully created auction with id {1}", Id, auctionId);* return "AuctioneerVIP " + Id + " successfully created auction with id " + auctionId;  
 }  
 }  
  
 return "An error occured during creating new auction";  
}

…

public override string CreateAuction(string auctionName, int initialValue, int auctionTime,  
 AuctionList<int> auctionList)  
{  
 var auctionId = auctionList.CreateAuction(auctionName, initialValue, auctionTime, Id);  
 if (auctionId != -1)  
 {  
 *//Console.WriteLine("AuctioneerVIP {0} successfully created auction with id {1}", Id, auctionId);* return "AuctioneerVIP " + Id + " successfully created auction with id " + auctionId;  
 }  
   
 return "An error occured during creating new auction";   
}

### Typy wyliczeniowe

Typ wyliczeniowy „ClientActions” przechowuje słowa klucze które będą wysyłane do serwera jako identyfikator typu zadania, jakie serwer ma zrealizować.

Prezentacja typu wyliczeniowego:

public enum ClientActions  
{  
 **Create**,  
 **ShowAuctions**,  
 **Bid**,  
 **AddFunds**,  
 **Quit**,  
 **Error**}

Wykorzystanie typu wyliczeniowego:

newAuction.Type = ClientActions.Create.ToString();

…

order.Type = ClientActions.**ShowAuctions**.ToString();

### Typy generyczne

Metoda „AuctionList” zawiera w swojej deklaracji „<T>”. Jest to deklaracja typu generycznego, na podstawie którego będziemy określać jakiego typu będzie identyfikator aukcji. W momencie deklaracji listy aukcji należy podać konkretny typ, na podstawie którego będzie deklarowany typ atrybutów dla klasy.

W ramach projektu implementacja wymusza zastosowanie typu „int”, ponieważ w dalszej kolejności służy to do tworzenia identyfikatora aukcji, który już nie jest typem generycznym. Nie mniej jednak umożliwia to szybkie dostosowanie implementacji w przypadku zmiany typu identyfikatora (wystarczyłaby zmiana deklaracji typu identyfikatora oraz mechanizmu jego inkrementacji).

Prezentacja deklaracji i wykorzystania typu generycznego

public class AuctionList<T>  
{  
 private T \_idCounter;  
 private readonly T \_idStep;  
 private List<Auction> \_auctionList = new();  
  
 public AuctionList(T auctionListIdCounter, T idStep)  
 {  
 \_idCounter = auctionListIdCounter;  
 \_idStep = idStep;  
 }

…

var auction = new Auction(Convert.ToInt32(\_idCounter), auctionName, initialValue, auctionTime, ownerId);  
  
if (auction.Id == -1) return -1;  
  
\_auctionList.Add(auction);  
  
var x = Expression.Parameter(typeof(T), "\_idCounter");  
var y = Expression.Parameter(typeof(T), "\_idStep");  
var body = Expression.Add(x, y);  
var incrementIdCounter = Expression.Lambda<Func<T, T, T>>(body, x, y).Compile();  
\_idCounter = incrementIdCounter(\_idCounter, \_idStep);

Prezentacja tworzenia listy aukcji z typem generycznym

\_auctionList = new AuctionList<int>(1, 1);

### Atrybuty

Zadeklarowane klasy posiadają swoje atrybuty przechowujące kluczowe informacje o klientach czy aukcjach. Atrybuty posiadają różne deklaracje, niektóre są typu „private” niedostępne poza klasą, „protected” niedostępne ogólnie, ale możliwe do użycia dla podklas, oraz „public” bez żadnej ochrony.

Atrybuty klasy „Client”, ich deklaracja oraz uzupełnianie w konstruktorze

public sealed class Auction : IValidateName, IDisposable  
{  
 public readonly int Id;  
 public readonly string Name;  
 public int Cost;  
 public readonly int OwnerId;  
 public int? WinnerId;  
 public DateTime TimeToEnd;  
   
 private bool \_disposed;  
  
 public Auction(int id, string name, int cost, int auctionTime, int ownerId)  
 {  
 if (!ValidateName(name.Trim()))  
 {  
 Console.WriteLine("Error, wrong auction name input ({0})", name);  
 Id = -1;  
 }  
 else  
 {  
 Id = id;  
 Name = Regex.Replace(name.Trim(), @"\s+", " ");  
 Cost = cost;  
 var warsawTimeZone = TimeZoneInfo.FindSystemTimeZoneById("Central European Standard Time");  
 var currentTimeInWarsaw = TimeZoneInfo.ConvertTimeBySystemTimeZoneId(DateTime.Now, warsawTimeZone.Id);  
  
 TimeToEnd = currentTimeInWarsaw.AddMinutes(auctionTime);  
 OwnerId = ownerId;  
 }  
 }

Atrybuty klasy „Auction”, ich deklaracja oraz uzupełnianie w konstruktorze

public abstract class Client  
{  
 protected readonly int Id;  
 private int \_money;  
 private static int \_idCounter = 1;  
 private readonly string \_name;  
  
 protected Client(int initMoney, string name)  
 {  
 Id = \_idCounter++;  
 \_money = initMoney;  
 \_name = name;  
 }

### Metody anonimowe, wyrażenia Lambda

W metodzie „CreateAuction” klasy „AuctionList” każda nowa aukcja tworzona jest z nowym, unikatowym identyfikatorem. Jest ona typem generycznym, stąd wykorzystuję wyrażenie Lambda oraz modułu „System.Linq.Expressions” do stałego zwiększania zmiennej „\_idCounter” o wartość „\_idStep” (obie wartości wstępnie ustalane w konstruktorze). Wyrażenie pozwala na dokonanie operacji sumy na tych dwóch zmiennych w taki sposób, aby program był w stanie obsłużyć typ generyczny.

Wykorzystanie wyrażenia lambda

var x = Expression.Parameter(typeof(T), "\_idCounter");  
var y = Expression.Parameter(typeof(T), "\_idStep");  
var body = Expression.Add(x, y);  
var incrementIdCounter = Expression.Lambda<Func<T, T, T>>(body, x, y).Compile();  
\_idCounter = incrementIdCounter(\_idCounter, \_idStep);

### Language Integrated Query (LINQ)

Wykorzystane są do przeszukiwania list aukcji oraz list klientów przy pomocy dwóch metod: „FirstOrDefault” oraz „Find”. Dla obu należy zadeklarować warunek, na podstawie którego przeszuka on listę. Dla identyfikatora aukcji będzie to warunek sprawdzający, czy ten identyfikator jest równy identyfikatorowi, którego szukamy. Dodatkowo dla metody „StringRepresentation” w ramach pliku „ExtensionMethods.cs” użyta jest metoda „Select”, która odwzorowuje dla każdego elementu listy jej reprezentację w formie tekstu.

Prezentacja użycia FirstOrDefault

var auctionToRemove = \_auctionList.FirstOrDefault(a => a.Id == auctionId);

Prezentacja użycia Find

clientsList.Find(c => c.GetId() == auctionToRemove.OwnerId)?.UpdateMoney('+', auctionToRemove.Cost);

Prezentacja użycia Select

return auctions.Select(a => $"{a.Id,-2} {a.Name,-10} {a.OwnerId,-2} {a.WinnerId,-2} {a.Cost,-5} {a.TimeToEnd}").ToList();

### Metody async, synchronizacja await lub Task API

Komunikacja ze strony serwera odbywa się w iteracjach. Jest tworzony i uruchamiany Task wykonujący asynchroniczną metodę w której następuje sekwencja odebrania wiadomości od klienta, przetworzenia zlecenia i wysłania odpowiedzi z powrotem do klienta. Po zakończeniu taska tworzony jest kolejny task wykonujący tę samą sekwencję działań.

Przykładowe zastosowania

private async void InitPipe()  
{  
 \_underlyingPipe = new NamedPipeServerStream(\_pipeName, PipeDirection.**InOut**);  
 await \_underlyingPipe.WaitForConnectionAsync();  
}  
  
public void WaitConnection()  
{  
 try  
 {  
 if (!\_underlyingPipe.IsConnected)  
 {  
 \_underlyingPipe.WaitForConnection();  
 }  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e);  
 throw;  
 }  
}

…

private async Task Communicate(PipeServer pipe)  
{  
 if (pipe.IsConnected() && !pipe.ToDelete)  
 {  
 pipe.WaitConnection();  
 var msg = pipe.Read();  
 var message = new Response();  
   
 var rec = JsonSerializer.Deserialize<Base>(msg);  
  
 if (rec != null)  
 switch (rec.Type)  
 {  
 case "Create":  
 var received = JsonSerializer.Deserialize<CreateAuction>(msg);  
 if (received != null)  
 message.Message = \_clientsList.Find(c => c.GetName() == pipe.GetName()).CreateAuction(  
 received.Name, received.Value, received.Time, \_auctionList);  
 *//snd = "Auction Created"; ThingsForAuction.Camera.ToString()  
 //message.message = "Auction Created";* break;  
 case "ShowAuctions":  
 *//var received2 = JsonSerializer.Deserialize<ShowAuctions>(msg);* var send = \_auctionList.PrintAllAuctions();  
 var stringList = JsonSerializer.Serialize(send);  
 message.Message = send.Any() ? "list" : "No auctions currently running!";  
  
 message.AuctionList = stringList;  
 *//Console.WriteLine(message.auctionList);  
 //snd = JsonSerializer.Serialize(send);* break;  
 case "Bid":  
 var received3 = JsonSerializer.Deserialize<BidAuction>(msg);  
 if (received3 != null)  
 message.Message = \_clientsList.Find(c => c.GetName() == pipe.GetName())  
 .Bid(received3.auctionId, received3.bidValue, \_auctionList, \_clientsList);  
 break;  
 case "AddFunds":  
 var received4 = JsonSerializer.Deserialize<Fund>(msg);  
 if (received4 != null)  
 message.Message = \_clientsList.Find(c => c.GetName() == pipe.GetName())  
 .UpdateMoney('+', received4.value);  
 break;  
 case "Quit":  
 *//pipe.close();* pipe.ToDelete = true;  
 message.Message = "Disconnected";  
 break;  
 }  
 pipe.WriteIfConnected(JsonSerializer.Serialize(message));  
 }

}

…

private void ClientCommunication()  
{  
 var toDelete = new List<PrivPipe>();  
 while (true)  
 {  
 \_auctionList.CloseAuctions(\_clientsList);  
 foreach (var pipeServer in \_pipeServers)  
 {  
 if (!pipeServer.IsCreated)  
 {  
 pipeServer.Task = new Task(() =>  
 {  
 Communicate(pipeServer.Pipe);  
 });  
   
 pipeServer.Task.Start();  
 pipeServer.IsCreated = true;  
 }  
 else if (pipeServer.Task.Status != TaskStatus.**Running** && pipeServer.Task.Status != TaskStatus.**WaitingForActivation** && pipeServer.Task.Status != TaskStatus.**WaitingToRun**)  
 {  
 pipeServer.Task.Dispose();  
 if (pipeServer.Pipe.IsConnected() &&  
 !pipeServer.Pipe.ToDelete)  
 {  
 pipeServer.Task = new Task(() =>  
 {  
 Communicate(pipeServer.Pipe);  
 });  
   
 pipeServer.Task.Start();  
 }  
 else if(pipeServer.Pipe.ToDelete)  
 {  
 toDelete.Add(pipeServer);  
 }  
   
 }  
 }  
  
 foreach (var privPipe in toDelete)  
 {  
 \_pipeServers.Remove(privPipe);  
 privPipe.Dispose();  
 }  
 toDelete.Clear();  
  
 Thread.Sleep(500);  
 }  
}

### Wykorzystanie wyrażeń regularnych

W momencie tworzenia nowej aukcji w konstruktorze klasy „Auction” sprawdzana jest z wykorzystaniem wyrażeń regularnych żądana nazwa aukcji. Jeżeli nazwa nie jest poprawnie zdefiniowana, aukcja nie jest tworzona. Poprawnie zdefiniowana nazwa to taka, w której każde jej słowo ma tylko pierwszą wielką literę.

Wyrażenie regularne zamieniające kilka spacji na jedną

Name = Regex.Replace(name.Trim(), @"\s+", " ");

Wyrażenie regularne sprawdzające, czy wszystkie słowa z tekście zaczynają się na wielką literę

var regex = new Regex("^[A-Z][a-z]+\\b(?:\\s+[A-Z][a-z]+\\b)\*$");  
return regex.IsMatch(name);

### Implementacja i prawidłowe wykorzystanie interfejsu IDisposable

Klasa „privPipe” zawiera atrybut klasy „pipeServer” który to z kolei zawiera atrybut typu „NamedPipeServerStream”. Ostatni atrybut implementuje interfejs „IDisposable” co powoduje, że każda następna klasa również implementuje go jak i jego metodę „Dispose()”. „privPipe” wywołuje tę metodę w destruktorze co powoduje kaskadowe wywołanie metody „Dispose()”, aż do obiektu „NamedPipeServerStream”.

Implementacje metody „Dispose()” dla połączeń z klientami

public void Dispose()  
{  
 try  
 {  
 Pipe.Dispose();  
 Task.Dispose();  
 }  
 catch (Exception ex)  
 {  
 Console.WriteLine(ex);  
 }  
}

…

public void Dispose()  
{  
 try  
 {  
 \_underlyingPipe.Close();  
 \_underlyingPipe.Dispose();  
 }  
 catch (Exception ex)  
 {  
 Console.WriteLine(ex);  
 }  
}

Użycie metody „Dispose()” w dekonstruktorze

~PrivPipe()  
{  
 Dispose();  
}

Użycie metody „Dispose()” dla Taska

\_pipeServer.Dispose();

### Extension method dla typu Base Class Library

W pliku „ExtensionMethods.cs” zawarta jest metoda „StringRepresentation” jako zupełnie nowa metoda dla listy. Metoda przyjmuje listę aukcji, a następnie za pomocą metody „Select” tłumaczy każdy jej element na jej reprezentację w formie tekstu. Na końcu scala te reprezentacje do listy typu „string”.

Implementacja metody

public static List<string> StringRepresentation(this List<Auction> auctions)  
{  
 return auctions.Select(a => $"{a.Id,-2} {a.Name,-10} {a.OwnerId,-2} {a.WinnerId,-2} {a.Cost,-5} {a.TimeToEnd}").ToList();  
}

Wykorzystanie metody

var auctionStrings = \_auctionList.StringRepresentation();

# Prezentacja w pełni automatycznej komunikacji międzyprocesowej

Zlecany jest Task do komunikacji z klientem. Jeżeli nie jest klient podłączony do danego pipe’a, czekamy, aż się połączy. Po podłączeniu serwer czeka aż pojawi się komunikat w pipe’ie odczytuje go i po przetworzeniu go od razu wystawia informację zwrotną do pipe’a.

# Wnioski

W projekcie udało się zastosować wszystkie niezbędne i kilka dodatkowych elementów z karty Scorecard.

Aplikacja zawiera dwustronną komunikację między procesami

Projekt był dobrym sposobem na przetrenowanie umiejętności programowania w grupie, dzielenia się pracą. Rozbieżności koncepcyjne wymagały stałej komunikacji pomiędzy członkami grupy co na początku nie sprawiało wrażenia problemu. Przed laboratoriami, żaden z nas nie znał .NET. Była to więc dobra okazja do nauki nowej technologii.