

# **Metody programowania .NET**

Sprawozdanie z Projektu z ćwiczeń laboratoryjnych

**Temat Projektu: Aplikacja do sprawdzania czasu  
przyjazdu wybranej komunikacji miejskiej**

**Autorzy: Mateusz Gajda, Kuba Kaczmarek**

**Grupa: WCY20IJ1S1**

**Prowadzący: Mgr inż. Krzysztof Mierzejewski**

# 1. Ogólny opis zaimplementowanego rozwiązania / use case.

## - Krótki opis działania programu

Aplikacja działa na dwóch oddzielnych procesach komunikujących się za pomocą plików. Pierwszy proces to proces pasażera - symuluje np. oczekującego na przystanku użytkownika, który wyciąga telefon i włącza aplikację aby sprawdzić za ile przyjedzie jego linia środka komunikacji, a drugi to proces systemu komunikacji miejskiej – symuluje on system, który odbiera z procesu pasażera odpowiednie informacje aby obliczyć, w zależności od napotkanego losowego opóźnienia, i przekazać z powrotem do użytkownika za ile przyjedzie wybrany przez niego środek transportu. Każdy środek transportu porusza się w pętli – z jednego przystanku granicznego na drugi, po czym wraca tą samą trasą z tymi samymi przystankami. Po drodze każdego środka transportu mogą natrafić się losowe zdarzenia powodujące opóźnienie pojazdu.

Działanie procesu użytkownika:

Użytkownik po uruchomieniu aplikacji ma wybór czy chce rozpocząć działanie czy wyjść z aplikacji – funkcja ta działa na pętli i po każdym zakończonym sprawdzeniu za ile przyjedzie środek transportu użytkownik wraca do momentu startowego w którym może ponownie wybrać czy chce sprawdzić za ile przyjedzie autobus czy wyjść z aplikacji. Po wybraniu opcji sprawdzenia za ile przyjedzie środek transportu program prosi użytkownika o niezbędne dane: środek transportu, którym chce jechać z podanej listy, linia danego środka transportu która go interesuje, przystanek na którym się obecnie oraz kierunek w którym chce jechać. Następnie dane te są przesyłane do procesu systemu komunikacji za pomocą funkcji Sender(). Następnie po otrzymaniu danych zwrotnych(funkcja Getter()) z procesu systemu komunikacji(czyli obliczonego czasu przyjazdu) proces ten wyświetla informacje za ile przyjedzie wybrana linia wybranego środka komunikacji wraz z uwzględnieniem wszelkich opóźnień. Z kolei po wybraniu opcji zamknięcia aplikacji proces ten wysyła za pomocą plików informacje do drugiego procesu żeby zamknął wszystkie działające Taski i zwraca wartość 0 oraz usuwa zbędne pliki UserData(funkcja Dispose()).

Działanie Procesu systemu komunikacji:

Po uruchomieniu procesu cztery obiekty klas środków transportu asynchronicznie pobierają swój rozkład jazdy każdej linii z odpowiedniego pliku(np. Autobus pobiera dane z pliku Bus.txt). Następnie proces ten czeka na to aż pojawi się odpowiedni plik(funkcja Getter()) od procesu użytkownika. Po pojawieniu się pliku automatycznie jest on odczytywany asynchronicznie przez cztery obiekty klas środków transportu po czym na podstawie tych danych wraz z uwzględnieniem losowych opóźnień jest obliczane za ile się powinna zjawić wybrana linia wybranego środka transportu na podanym przez użytkownika przystanku(funkcja CalculationOfDelay()). Pierwsza część obliczania opóźnienia polega na wyliczeniu aktualnego przystanku na którym znajduje się pojazd, uwzględniając losowe opóźnienia. W drugiej części wyliczany jest czas w minutach potrzebny na dojechanie do przystanku użytkownika w wybranym kierunku. Następnie obliczony czas jest przesyłany przy użyciu pliku z powrotem do procesu użytkownika(funkcja Sender()). Proces systemu komunikacji po wszystkich działaniach oczekuje na dalsze działania użytkownika – jeśli

użytkownik zdecyduje się zamknąć aplikację to proces ten również się zamknie poprzez odczytanie plików End(działa to na zasadzie komunikacji międzyprocesowej przy użyciu plików) i usunie zbędne pliki(funkcja Dispose()), natomiast jeśli zdecyduje się dalej sprawdzać za ile dana linia przyjedzie to proces ten będzie oczekiwał na odpowiedni plik.

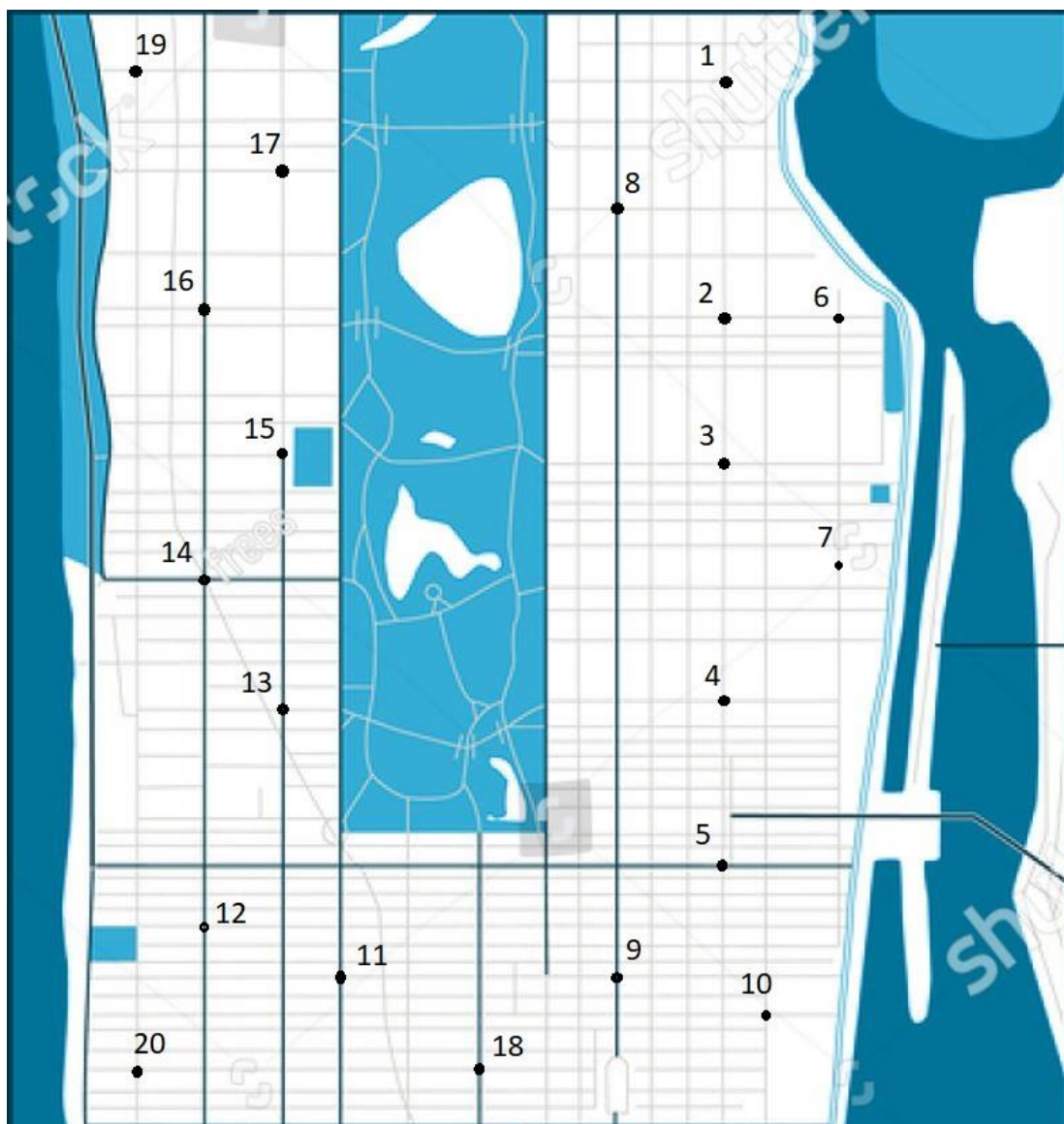
Cały program opiera się o wymyślonych przez nas danych, które będą przedstawione poniżej opisu. Dodatkowo **cały program pisaliśmy razem** i nie dzieliliśmy się na procesy – staraliśmy się wszystko wspólnie przeanalizować i wypełnić przedstawiony problem razem.

### - Wymyślone dane

- Tabela ze współczynnikami prawdopodobieństwa wystąpienia danego opóźnienia i jak ono wpływa na czas przyjazdu danego środka transportu (prawdopodobieństwo x i y jest różne dla każdego środka transportu)

Pojazd	szansa wystąpienia opóźnienia	Praw x	Praw y	w. godz szczytu	w. nie godz szczytu
		Małe opóźnienie	Potężne Opóźnienie		
A	0,5	1,20	1,50	1,70	1,30
T	0,25	1,20	1,70	1,50	1,20
M	0,1	1,10	2,30	1,00	1,00
P	0,15	1,20	2,00	1,00	1,00

- Rysunek mapy wymyślonej komunikacji miejskiej z zaznaczonymi przystankami – rozmieszczenie przystanków jest poglądowe i pokazuje odległości między nimi, a co za tym idzie czas przyjazdu z jednego przystanku do drugiego



- Pliki tekstowe z rozkładem jazdy poszczególnych środków transportu z zawartymi danymi takimi jak: numer linii, przystanek i planowy czas przyjazdu z jednego przystanku na drugi bez opóźnień

- Bus.txt

A1

1: 0

2: 5

3: 4

4: 5

5: 5

9: 4

8: 15

=

A2

19: 0

17: 5

16: 3

15: 3

14: 4

=

A3

14: 0

13: 5

12: 8

20: 4

17: 7

=

A4

20: 0

12: 4

11: 3

18: 5

9: 6

5: 4

10: 6

=

A5

1: 0

5: 19

18: 10

12: 8

19: 20

=

A6

10: 0

5: 6

4: 5

7: 4

6: 4

1: 8

=

A7

2: 0

6: 3

3: 6

7: 4

4: 4

=

A8

14: 0

11: 16

18: 5

10: 9

7: 10

=

A9

20: 0

13: 12

18: 10

4: 15

10: 11

=

A10

8: 0

7: 10

9: 10

12: 6

20: 4

=

- Metro.txt

M1

12: 0

14: 2

16: 1

19: 1

=

- Train.txt

P1

19: 0

12: 4

5: 4

1: 4

=

- Tram.txt

T1

19: 0

16: 4

14: 3

13: 2

=

T2

10: 0

5: 3

11: 6

12: 2

20: 2

=

T3

9: 0

4: 4

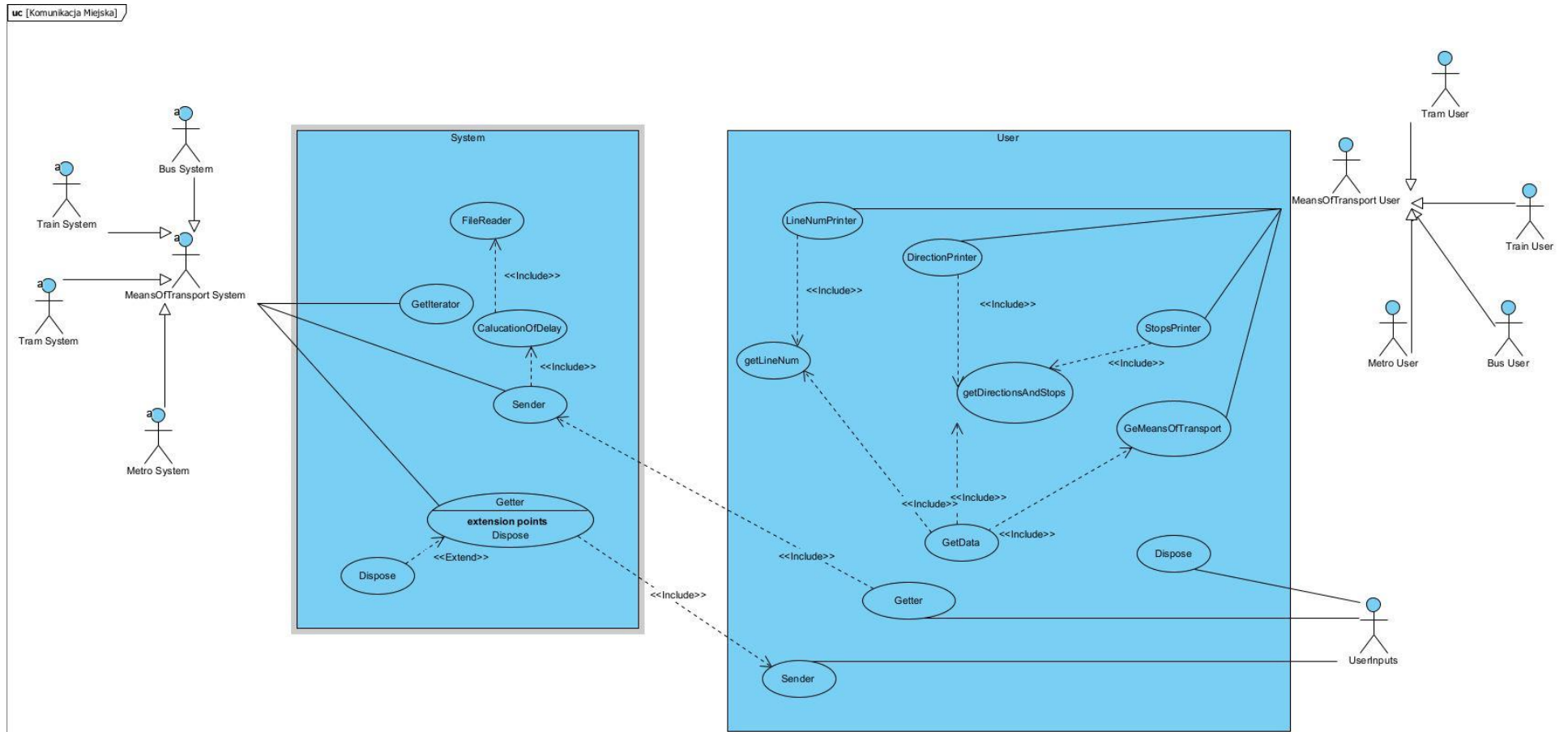
7: 3

8: 5

1: 2

=

uc [Komunikacja Miejska]



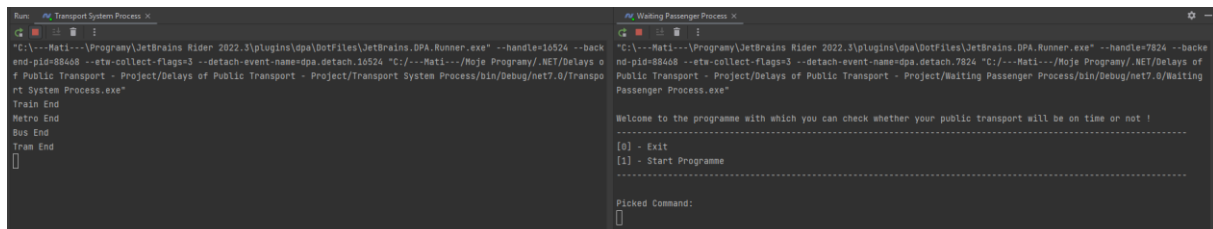


## 2. Specyficzne wymagania odnośnie konfiguracji środowiska w celu uruchomienia i dynamicznej oceny rozwiązania.

Szczególnym wymaganiem odnośnie konfiguracji środowiska w celu uruchomienia i dynamicznej oceny rozwiązania jest to, że lokalizacja plików .txt powinna zostać w domyślnym folderze projektu. Dodatkowo korzystaliśmy z wersji .NET 6.0 i SDK 6.

## 3. Opis demonstracyjnej ścieżki uruchomienia rozwiązania.

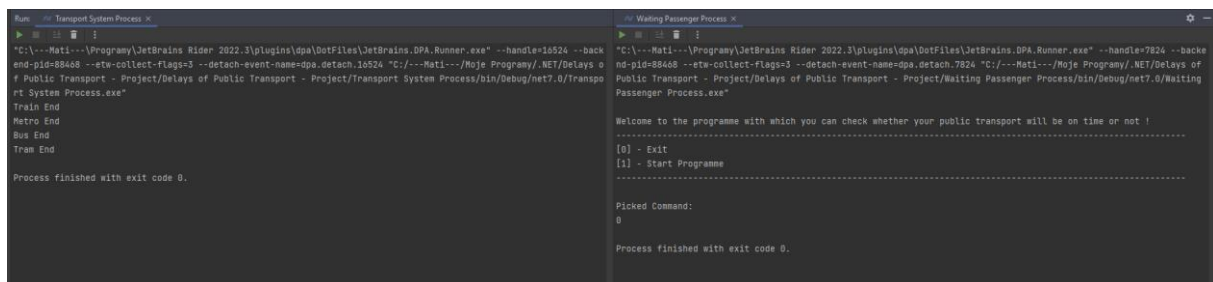
### 1. Uruchomienie aplikacji



```
Run: Transport System Process x
"C:\---Mati---\Programy\JetBrains Rider 2022.3\plugins\opa\DotFiles\JetBrains.OPA.Runner.exe" --handle=16524 --back
end-pid=88468 --etw-collect-flags=3 --detach-event-name=opa.detach.16524 "C:/---Mati---/Moje Programy/.NET/Delays o
f Public Transport - Project/Delays of Public Transport - Project/Transport System Process/bin/Debug/net7.0/Transpo
rt System Process.exe"
Train End
Metro End
Bus End
Tram End
[]

Waiting Passenger Process x
"C:\---Mati---\Programy\JetBrains Rider 2022.3\plugins\opa\DotFiles\JetBrains.OPA.Runner.exe" --handle=7824 --backe
nd-pid=88468 --etw-collect-flags=3 --detach-event-name=opa.detach.7824 "C:/---Mati---/Moje Programy/.NET/Delays of
Public Transport - Project/Delays of Public Transport - Project/Waiting Passenger Process/bin/Debug/net7.0/Waiting
Passenger Process.exe"
Welcome to the programme with which you can check whether your public transport will be on time or not !
-----
[0] - Exit
[1] - Start Programme
-----
Picked Command:
[]
```

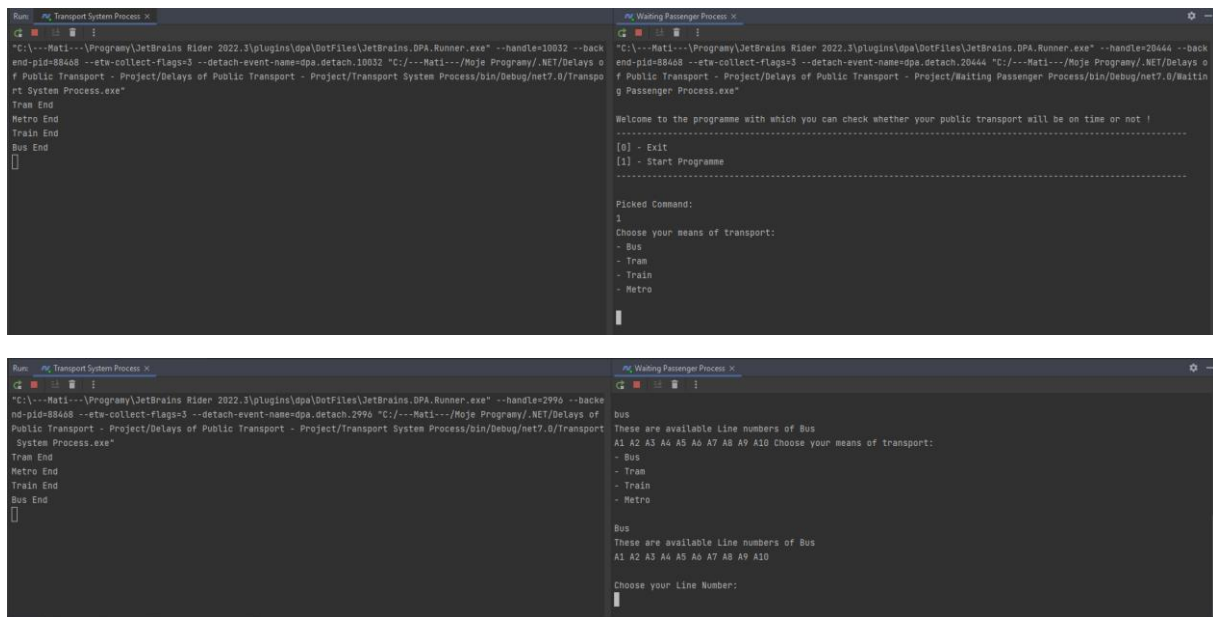
### 2. Wybranie zamknięcia aplikacji



```
Run: Transport System Process x
"C:\---Mati---\Programy\JetBrains Rider 2022.3\plugins\opa\DotFiles\JetBrains.OPA.Runner.exe" --handle=16524 --back
end-pid=88468 --etw-collect-flags=3 --detach-event-name=opa.detach.16524 "C:/---Mati---/Moje Programy/.NET/Delays o
f Public Transport - Project/Delays of Public Transport - Project/Transport System Process/bin/Debug/net7.0/Transpo
rt System Process.exe"
Train End
Metro End
Bus End
Tram End
Process finished with exit code 0.

Waiting Passenger Process x
"C:\---Mati---\Programy\JetBrains Rider 2022.3\plugins\opa\DotFiles\JetBrains.OPA.Runner.exe" --handle=7824 --backe
nd-pid=88468 --etw-collect-flags=3 --detach-event-name=opa.detach.7824 "C:/---Mati---/Moje Programy/.NET/Delays of
Public Transport - Project/Delays of Public Transport - Project/Waiting Passenger Process/bin/Debug/net7.0/Waiting
Passenger Process.exe"
Welcome to the programme with which you can check whether your public transport will be on time or not !
-----
[0] - Exit
[1] - Start Programme
-----
Picked Command:
0
Process finished with exit code 0.
```

### 3. Wybranie Rozpoczęcia działania aplikacji – kolejne etapy działania aplikacji



```
Run: Transport System Process x
"C:\---Mati---\Programy\JetBrains Rider 2022.3\plugins\opa\DotFiles\JetBrains.OPA.Runner.exe" --handle=10032 --back
end-pid=88468 --etw-collect-flags=3 --detach-event-name=opa.detach.10032 "C:/---Mati---/Moje Programy/.NET/Delays o
f Public Transport - Project/Delays of Public Transport - Project/Transport System Process/bin/Debug/net7.0/Transpo
rt System Process.exe"
Train End
Metro End
Tram End
Bus End
[]

Waiting Passenger Process x
"C:\---Mati---\Programy\JetBrains Rider 2022.3\plugins\opa\DotFiles\JetBrains.OPA.Runner.exe" --handle=20444 --back
end-pid=88468 --etw-collect-flags=3 --detach-event-name=opa.detach.20444 "C:/---Mati---/Moje Programy/.NET/Delays o
f Public Transport - Project/Delays of Public Transport - Project/Waiting Passenger Process/bin/Debug/net7.0/Waitin
g Passenger Process.exe"
Welcome to the programme with which you can check whether your public transport will be on time or not !
-----
[0] - Exit
[1] - Start Programme
-----
Picked Command:
1
Choose your means of transport:
- Bus
- Tram
- Train
- Metro
[]

Run: Transport System Process x
"C:\---Mati---\Programy\JetBrains Rider 2022.3\plugins\opa\DotFiles\JetBrains.OPA.Runner.exe" --handle=2996 --backe
nd-pid=88468 --etw-collect-flags=3 --detach-event-name=opa.detach.2996 "C:/---Mati---/Moje Programy/.NET/Delays o
f Public Transport - Project/Delays of Public Transport - Project/Transport System Process/bin/Debug/net7.0/Transpo
rt System Process.exe"
Train End
Metro End
Tram End
Bus End
[]

Waiting Passenger Process x
"C:\---Mati---\Programy\JetBrains Rider 2022.3\plugins\opa\DotFiles\JetBrains.OPA.Runner.exe" --handle=20444 --back
end-pid=88468 --etw-collect-flags=3 --detach-event-name=opa.detach.20444 "C:/---Mati---/Moje Programy/.NET/Delays o
f Public Transport - Project/Delays of Public Transport - Project/Waiting Passenger Process/bin/Debug/net7.0/Waitin
g Passenger Process.exe"
Welcome to the programme with which you can check whether your public transport will be on time or not !
-----
[0] - Exit
[1] - Start Programme
-----
Picked Command:
1
Choose your means of transport:
- Bus
- Tram
- Train
- Metro
[]

Bus
These are available Line numbers of Bus
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 Choose your means of transport:
- Bus
- Tram
- Train
- Metro
[]

Bus
These are available Line numbers of Bus
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10
Choose your Line Number:
[]
```

```
Run: Transport System Process X
"C:\---Mati---\Programy\JetBrains Rider 2022.3\plugins\opa\DotFiles\JetBrains.OPA.Runner.exe" --handle=2996 --backe
nd-pid=88468 --eta-collect-flags=3 --detach-event-name=opa.detach.2996 "C:/---Mati---/Moje Programy/.NET/Delays of
Public Transport - Project/Delays of Public Transport - Project/Transport System Process/bin/Debug/net7.0/Transport
System Process.exe"
Tram End
Metro End
Train End
Bus End
[]

Waiting Passenger Process X
- Tram
- Train
- Metro

Bus
These are available Line numbers of Bus
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10

Choose your Line Number:
A5
These are available Stops of A5 Line
1 5 18 12 19

Choose your stop:
18
```

```
Run: Transport System Process X
"C:\---Mati---\Programy\JetBrains Rider 2022.3\plugins\opa\DotFiles\JetBrains.OPA.Runner.exe" --handle=2996 --backe
nd-pid=88468 --eta-collect-flags=3 --detach-event-name=opa.detach.2996 "C:/---Mati---/Moje Programy/.NET/Delays of
Public Transport - Project/Delays of Public Transport - Project/Transport System Process/bin/Debug/net7.0/Transport
System Process.exe"
Tram End
Metro End
Train End
Bus End
[]

Waiting Passenger Process X
Bus
These are available Line numbers of Bus
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10

Choose your Line Number:
A5
These are available Stops of A5 Line
1 5 18 12 19

Choose your stop:
18
These are available Directions of A5 Line:
* End Stop #1: 1
* End Stop #2: 19
```

```
Run: Transport System Process X
Public Transport - Project/Delays of Public Transport - Project/Transport System Process/bin/Debug/net7.0/Transport
System Process.exe"
Tram End
Metro End
Train End
Bus End
Calculating Delay...
Line: A5
Destination Stop: 18
Waiting Time: 66
Current Stop of Transport: 5
Current Direction of Transport: 19
[]

Waiting Passenger Process X
These are available Stops of A5 Line
1 5 18 12 19

Choose your stop:
18
These are available Directions of A5 Line:
* End Stop #1: 1
* End Stop #2: 19
1
It's 6:20
Your BUS A5 is currently at the 5 and will arrive in 66 min
```

#### 4. Ponowne wybranie wyjścia z aplikacji

```
Run: Transport System Process X
System Process.exe"
Tram End
Metro End
Train End
Bus End
Calculating Delay...
Line: A5
Destination Stop: 18
Waiting Time: 66
Current Stop of Transport: 5
Current Direction of Transport: 19
Process finished with exit code 0.

Waiting Passenger Process X
Welcome to the programme with which you can check whether your public transport will be on time or not !
[0] - Exit
[1] - Start Programme

Picked Command:
0
UserData.txt deleted
Process finished with exit code 0.
```

Wybrana ścieżka przedstawia sprawdzenie za ile przyjedzie autobus linii A5 będąc na przystanku 18 i chcąc jechać w kierunku przystanku 1. Informacja wyświetlona ukazuje wylosowaną godzinę sprawdzania przez użytkownika i to że ten autobus przyjedzie za 66 min. Jakby użytkownik chciał jeszcze raz sprawdzić inny środek transportu albo inną linię autobusu może wybrać ponownie opcje 1 zamiast opcji wyjścia z aplikacji.

#### 4. Dla każdego z projektów osobno: wskazanie w kodzie źródłowym wraz z krótkim uzasadnieniem dla zastosowania mechanizmu każdego pokrytego ze Scorecard punktu.

##### Scorecard

P1 - Passenger Process

P2 - System Process

##### Ocena db.

P1	P2
+	<input type="checkbox"/> Dwustronna komunikacja międzyprocesowa <b>Programowanie obiektowe</b> <input type="checkbox"/> Kontrakty: klasy abstrakcyjne, interfejsy <input type="checkbox"/> Polimorfizm: uogólnianie typów, przesłanianie metod <input type="checkbox"/> Typy wyliczeniowe <input type="checkbox"/> Zaprojektowanie typów generycznych, kowariancja i kontrawariancja <b>Metadane</b> <input type="checkbox"/> Atrybuty: własne lub wykorzystanie istniejących <b>Programowanie funkcyjne</b> <input type="checkbox"/> Delegaty: metody anonimowe, wyrażenia lambda <input type="checkbox"/> Kolekcje danych i Language Integrated Query <b>Programowanie asynchroniczne</b> <input type="checkbox"/> Metody async, synchronizacja await lub Task API

##### + ½ do oceny

Przynajmniej 3 punkty z poniższej listy:

- + + 1. Wykorzystanie wyrażań regularnych
- + + 2. Implementacja i prawidłowe wykorzystanie interfejsu IDisposable
- + 3. Extension methods dla typów z Base Class Library
- + 4. Użycie CancellationToken
- + 5. Wykorzystanie synchronization primitives

##### + ½ do oceny

- + + • Komunikacja międzyprocesowa jest w pełni automatyczna, tj. nie wymaga ręcznego wyzwalania pobierania komunikatów po stronie odbiorcy.

##### - ½ do oceny

- Nie uwalnianie zasobów z interfejsem IDisposable, jeżeli jest on dostępny w wykorzystywanych obiektach.

Każdy Punkt z Scorecard jest wskazany w kodzie za pomocą komentarzy. Krótkie opisy i przykłady użycia tych punktów zostaną zawarte poniżej (niektóre podpunkty zostały przez nas wykorzystane kilka razy co sprowadza się do dużej ilości kodu która mogłaby być potem nie czytelna w sprawozdaniu).

#### - Waiting Passenger Process

- Dwustronna komunikacja międzyprocesowa (Jest ona w pełni automatyczna):

W tym przypadku w pliku Program.cs tworzymy plik końca pracy programu – wysyłana jest informacja do drugiego procesu o zakończeniu pracy w postaci 4 plików po jednym dla każdego środka transportu. Niestety nie udało się nam zaimplementować zaproponowanego przez Pana sposobu aktywnego oczekiwania. Staraliśmy się używać FileSystemWatcher, jednakże po jego implementacji program nie działał zgodnie z naszymi założeniami, dlatego uznaliśmy, że najlepszym rozwiązaniem będzie nie wprowadzanie zmian.

```
//Komunikat o wyłączeniu procesu - Punkt w Scorecard: Dwustronna
komunikacja międzyprocesowa
case "0":
    string EndDataB = $"{directory}EndDataA.txt";
    using (var writer = new StreamWriter(EndDataB))
    {
        writer.WriteLine("END");
        writer.Flush();
    }
    string EndDataM = $"{directory}EndDataM.txt";
    using (var writer = new StreamWriter(EndDataM))
    {
        writer.WriteLine("END");
        writer.Flush();
    }
    string EndDataP = $"{directory}EndDataP.txt";
    using (var writer = new StreamWriter(EndDataP))
    {
        writer.WriteLine("END");
        writer.Flush();
    }
    string EndDataT = $"{directory}EndDataT.txt";
    using (var writer = new StreamWriter(EndDataT))
    {
        writer.WriteLine("END");
        writer.Flush();
    }
}
```

W tym przypadku w pliku UserInput.cs tworzymy dwie funkcje obsługujące komunikację między procesową która wysyła dane do pliku i odczytuje dane z innego (odpowiednio Sender() i Getter()). Getter() oczekuje na pojawienie się odpowiedniego pliku i użytkownik nie musi nic robić po jego utworzeniu – program automatycznie odczyta z niego dane przy użyciu pętli while. Wywoływane są one w następujący sposób:

```
//Wywołanie metod odpowiadających za komunikację międzyprocesowa - Punkt w
Scorecard: Dwustronna komunikacja międzyprocesowa
Sender(UserMeansOfTransport, UserLineNum, UserStop, UserDirection, Hour,
Minutes);
Getter();
```

```

//Utworzenie metody odpowiadajacej za komunikacje miedzyprocesowa (wysyla
dane do drugiego procesu) - Punkt w Scorecard: Dwustronna komunikacja
miedzyprocesowa
void Getter()
{
    string errorFinder;
    string SystemData = $"{directory}SystemData.txt";
    while (!File.Exists(SystemData))
    {
        Thread.Sleep(1000);
    }
    using (var reader = new StreamReader(SystemData))
    {
        Console.WriteLine("-----");
        Console.WriteLine(Minutes > 9 ? $"It's {Hour}:{Minutes}" : $"It's
{Hour}:0{Minutes}");
        errorFinder = reader.ReadLine();
        if (errorFinder == "Error")
        {
            Console.WriteLine(reader.ReadLine());
        }
        else
        {
            CurrentStop = int.Parse(errorFinder);
            WaitingTime = double.Parse(reader.ReadLine());
            Console.WriteLine($"Your {UserMeansOfTransport} {UserLineNum}
is currently at the {CurrentStop} and will arrive in {WaitingTime} min");
            Console.WriteLine("-----\n");
        }
    }
    File.Delete(SystemData);
}

//Utworzenie metody odpowiadajacej za komunikacje miedzyprocesowa (odbiera
dane od drugiego procesu) - Punkt w Scorecard: Dwustronna komunikacja
miedzyprocesowa
void Sender(string UserMeansOfTransport, string UserLineNum, string
UserStop, string UserDirection, int Hour, int Minutes)
{
    string UserData = $"{directory}UserData{iterator}.txt";
    using (var writer = new StreamWriter(UserData))
    {
        writer.WriteLine(UserMeansOfTransport);
        writer.WriteLine(UserLineNum);
        writer.WriteLine(UserStop);
        writer.WriteLine(UserDirection);
        writer.WriteLine(Hour);
        writer.WriteLine(Minutes);
        writer.Flush();
    }
}

```

- Kontrakty: klasy abstrakcyjne, interfejsy:

Utworzony interface zawiera funkcje dla zaprojektowanego typu generycznego.

```
//Utworzenie Interface - Punkt w Scorecard: Kontrakty: Interface
public interface MyGenericListInterface<T>
{
    public void Add(T item);
    public void Remove(T item);
    public bool Contains(T item);
    public int Count();
    public void Dispose();
    public List<T> ReadFromFile(string path);
}
```

Wykorzystanie interface:

```
//Wykorzystanie Interface - Punkt w Scorecard: Kontrakty: Interface
class MyGenericList<T> : MyGenericListInterface<T>
{
    private List<T> _items = new List<T>();
    public void Add(T item)
    {
        _items.Add(item);
    }

    public void Remove(T item)
    {
        _items.Remove(item);
    }

    public bool Contains(T item)
    {
        return _items.Contains(item);
    }

    public int Count()
    {
        return _items.Count;
    }

    public void Dispose()
    {
        _items = null;
    }

    public List<T> ReadFromFile(string path)
    {
        using (var file = File.OpenText(path))
        {
            string line;
            while ((line = file.ReadLine()) != null)
            {
                T item = (T)Convert.ChangeType(line, typeof(T));
                _items.Add(item);
            }
        }
        return _items;
    }
}
```

- Polimorfizm: uogólnianie typów, przesłanianie metod

Utworzony plik MeansOfTransport.cs jest klasą macierzystą i dziedziczą po niej klasy Bus, Metro, Tram oraz Train. Klasy te nadpisują większość funkcji z klasy macierzystej.

- Typy wyliczeniowe

Głównym zastosowaniem typów wyliczeniowych w naszym programie jest iterowanie po listach podczas czytania pliku np.:

```
public override IEnumerable<string> getLineNum()
{
    var filelist = new MyGenericList<string>();
    IEnumerable<string> Lines = filelist.ReadFromFile(path).Where(L => L[0]
== parameters.meansOfTransportLetter);
    return Lines;
}
```

```
//Wykożystanie typow wyliczeniowych - Punkt w Scorecard: Typy wyliczeniowe
foreach (var LineNumber in getLineNum())
{
    Console.WriteLine($"{LineNumber} ");
}
```

- Zaprojektowanie typów generycznych, kowariancja i kontrawariancja

Zaprojektowany przez nas typ generyczny odpowiada liście, w którą wpisywana jest zawartość plików:

```
class MyGenericList<T> : MyGenericListInterface<T>
{
    private List<T> _items = new List<T>();
    public void Add(T item)
    {
        _items.Add(item);
    }

    public void Remove(T item)
    {
        _items.Remove(item);
    }

    public bool Contains(T item)
    {
        return _items.Contains(item);
    }

    public int Count()
    {
        return _items.Count;
    }

    public void Dispose()
    {
        _items = null;
    }
}
```

```

public List<T> ReadFromFile(string path)
{
    using (var file = File.OpenText(path))
    {
        string line;
        while ((line = file.ReadLine()) != null)
        {
            T item = (T)Convert.ChangeType(line, typeof(T));
            _items.Add(item);
        }
    }
    return _items;
}

```

- Atrybuty: własne lub wykorzystanie istniejących

Atrybuty powołane są w oddzielnej klasie Parameters.cs i dla każdego środka transportu są one wpisywane przy użyciu konstruktora przed klasą:

```

//Utworzenie parametrow dla klas - Punkt w Scorecard: Atrybuty własne
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]

public class Parameters : Attribute
{
    public string FileName { get; set; }
    public string meansOfTransport { get; set; }
    public char meansOfTransportLetter { get; set; }

    //Zdefiniowanie wzoru do czytania pliku - Punkt w Scorecard:
    Wykorzystanie wyrazen regularnych
    public string pattern { get; set; } = @"[0-9]{1,2}";

    public Parameters(string FileName, string meansOfTransport, char
meansOfTransportLetter)
    {
        this.FileName = FileName;
        this.meansOfTransport = meansOfTransport;
        this.meansOfTransportLetter = meansOfTransportLetter;
    }
}

```

```

//Powolanie parametrow dla klasy Bus - Punkt w Scorecard: Atrybuty własne
[Parameters("Bus.txt", "Bus", 'A')]

```

- Delegaty: metody anonimowe, wyrażenia lambda

Wyrażenia lambda w tym programie zostały głównie wykorzystane razem z LINQ przy wybieraniu interesujących nas fragmentów pliku np.:

```

IEnumerable<string> Lines = filelist.ReadFromFile(path).Where(L => L[0] ==
parameters.meansOfTransportLetter);

```



- Kolekcje danych i Language Integrated Query

LINQ w tym programie zostały głównie wykorzystane razem z wyrażeniami lambda przy wybieraniu interesujących nas fragmentów pliku np.:

```
IEnumerable<string> Lines = filelist.ReadFromFile(path).Where(L => L[0] == parameters.meansOfTransportLetter);
```

- Metody async, synchronizacja await lub Task API

Proces Waiting Passenger został zaprojektowany w sposób taki aby każdy jego etap był wykonywany sekwencyjnie, dlatego metody async zostały wykonane jedynie w drugim procesie projektu.

- Wykorzystanie wyrażeń regularnych

Wyrażenia regularne służą do zwięzłego wyszukiwania interesujących nas fragmentów linii pliku takich jak np. przystanki i czas przyjazdu między nimi. Wzorzec określa fragmenty stringów pobranych z pliku składających się z jednej lub dwóch cyfr:

```
//Zdefiniowanie wzoru do czytania pliku - Punkt w Scorecard: Wykorzystanie wyrażeń regularnych
public string pattern { get; set; } = @"[0-9]{1,2}";
```

```
//Wykorzystanie wyrażeń regularnych - Punkt w Scorecard: Wykorzystanie wyrażeń regularnych
MatchCollection matches = Regex.Matches(direction, parameters.pattern);
DirectionsAndStops.Add(matches[0].ToString());
```

- Implementacja i prawidłowe wykorzystanie interfejsu IDisposable

Zastosowanie interfejsu IDisposable pozwala na zdefiniowanie metody Dispose(), dzięki której jesteśmy w stanie pozbyć się nieużywanych już plików, służących do komunikacji międzyprocesowej(przekazywanie parametrów danej linii) – wykorzystany został w pliku UserInput.cs:

```
//Zdefiniowanie funkcji Dispose - Punkt w Scorecard: Implementacja
interface IDisposableable
{
    public void Dispose()
    {
        for (int i = 1; i <= iterator; i++)
        {
            File.Delete($"{directory}UserData{i}.txt");
            Console.WriteLine($"UserData{i}.txt deleted");
        }
    }
}
```

- Wykorzystanie synchronization primitives

Nawiązując do podpunktu „Metody async, synchronizacja await lub Task API” tutaj również nie posiadamy uzasadnionego zastosowania tego podpunktu w tym procesie – został on wykorzystany w drugim procesie.

## - Transport System Process

- Dwustronna komunikacja międzyprocesowa (Jest ona w pełni automatyczna):

W tym przypadku każdy plik do środka komunikacji miejskiej posiada swoje funkcje, które oczekują na pojawienie się odpowiednich plików, po czym jeśli się pojawią to wykonują określone czynności – albo program kończy działanie(pojawienie się pliku End), albo program czyta dane od użytkownika i przekazuje do obliczania czasu przyjazdu. Używane są do tego wcześniej opisane funkcje Getter() i Sender().:

```
//Utworzenie metody odpowiadającej za komunikację międzyprocesową (wysyła dane do drugiego procesu) - Punkt w Scorecard: Dwustronna komunikacja międzyprocesowa
//Asynchroniczne wywołanie funkcji Getter() - Punkt w Scorecard: Metody async
public async Task Getter()
{
    string UserData = $"{directory}UserData{iterator}.txt";
    string EndData = $"{directory}EndDataA.txt";

    //Asynchroniczne wywołanie funkcji Getter() - Punkt w Scorecard: Metody async
    await Task.Run(() =>
    {
        while (!File.Exists(UserData) && !File.Exists(EndData))
        {
            Thread.Sleep(1000);
        }

        if (File.Exists(UserData))
        {
            using (var reader = new StreamReader(UserData))
            {
                UserMeansOfTransport = reader.ReadLine();
                UserLineNum = reader.ReadLine();
                UserStop = reader.ReadLine();
                UserDirection = reader.ReadLine();
                UserHour = int.Parse(reader.ReadLine());
                UserMinutes = int.Parse(reader.ReadLine());
            }

            if (UserLineNum.First() == 'A')
            {
                End = false;
                CalculationOfDelay();
            }
        }
        else if (File.Exists(EndData))
        {
            End = true;

            //Wywołanie funkcji Dispose() - Punkt w Scorecard: Implementacja interfejsu IDisposable
            Dispose();
        }
    });
}

//Utworzenie metody odpowiadającej za komunikację międzyprocesową (odbiera
```

```

dane od drugiego procesu) - Punkt w Scorecard: Dwustronna komunikacja
miedzyprocesowa
public void Sender(string CurrentStop, double WaitingTime, bool toEarly)
{
    string SystemData = $"{directory}SystemData.txt";
    using (var writer = new StreamWriter(SystemData))
    {
        if(!toEarly)
        {
            writer.WriteLine(CurrentStop);
            writer.WriteLine(WaitingTime);
            writer.Flush();
        }
        else
        {
            writer.WriteLine("Error");
            writer.WriteLine($"It's too early for this line, first bus will
come at {parameters.StartHour_H}:{parameters.StartHour_Min}0");
            writer.Flush();
        }
    }
}

```

- Kontrakty: klasy abstrakcyjne, interfejsy:

Utworzony interfejs zawiera funkcje dla każdej klasy środka transportu:

```

//Utworzenie Interface - Punkt w Scorecard: Kontrakty: Interface
//Zastosowanie interface IDisposable - Punkt w Scorecard: Implementacja
interface'u IDisposable
public interface MeansOfTransport : IDisposable
{
    Task FileReader();
    void CalculationOfDelay();
    Task Getter();
    void Sender(string CurrentStop, double WaitingTime, bool toEarly);

    int getIterator { get; set; }
}

```

- Polimorfizm: uogólnianie typów, przesłanianie metod

Polimorfizm został użyty w tym przypadku dla typu generycznego w następujący sposób:

```

//Zaprojektowanie typu generycznego - Punkt w Scorecard: Zaprojektowanie
typow generycznych
//Polimorfizm, Dziedziczenie klas, stworzenie klasy maciezystej - Punkt w
Scorecard: Polimorfizm
public class MyGenericListPolymorphism<T>: List<T>
{
    private List<T> _items = new List<T>();
    public virtual List<T> getFileList(string path) { return _items;}
}

```

Po czym klasa MyGenericList z niej dziedziczy:

```
//Polimorfizm, Dziedziczenie klas - Punkt w Scorecard: Polimorfizm
//Zaprojektowanie typu generycznego - Punkt w Scorecard: Zaprojektowanie
typow generycznych
class MyGenericList<T> : MyGenericListPolymorphism<T>
{
    List<T> _items = new List<T>();
    public override List<T> getFileList(string path)
    {
        List<T> _items = new List<T>();
        using (var file = File.OpenText(path))
        {
            string line;
            while ((line = file.ReadLine()) != null)
            {
                T item = (T)Convert.ChangeType(line, typeof(T));
                _items.Add(item);
            }
        }
        return _items;
    }
}
```

- Typy wyliczeniowe

Głównym zastosowaniem typów wyliczeniowych w naszym programie jest iterowanie po listach podczas czytania pliku np.:

```
IEnumerable<string> Lines = filelist.getFileList(path).Where(L => L[0] == 'A');
```

- Zaprojektowanie typów generycznych, kowariancja i kontrawariancja

Zaprojektowany przez nas typ generyczny odpowiada liście, w którą wpisywana jest zawartość plików. Dodatkowo pozwala to na wykorzystanie kowariancji przy zwiększaniu wartości zmiennej iterator dla każdej klasy środka transportu:

```
//Polimorfizm, Dziedziczenie klas - Punkt w Scorecard: Polimorfizm
//Zaprojektowanie typu generycznego - Punkt w Scorecard: Zaprojektowanie
typow generycznych
class MyGenericList<T> : MyGenericListPolymorphism<T>
{
    List<T> _items = new List<T>();
    public override List<T> getFileList(string path)
    {
        List<T> _items = new List<T>();
        using (var file = File.OpenText(path))
        {
            string line;
            while ((line = file.ReadLine()) != null)
            {
                T item = (T)Convert.ChangeType(line, typeof(T));
                _items.Add(item);
            }
        }
        return _items;
    }
}
```

```
//Wykorzystanie typu generycznego i kowariancji - Punkt w Scorecard:
Zaprojektowanie typu generycznego i kowariancja
MyGenericList<MeansOfTransport> meansOfTransport = new
MyGenericList<MeansOfTransport>{Bus,Metro,Train,Tram};
while (!Bus.End && !Metro.End && !Train.End && !Tram.End)
{
    foreach (var meanOfTransport in meansOfTransport)
    {
        meanOfTransport.getIterator += 1;
    }

    //Oczekiwanie na zakończenie wszystkich asynchronicznych taskow - Punkt
w Scorecard: Metody async, synchronizacja await
    await Task.WhenAll(Bus.Getter(), Metro.Getter(), Train.Getter(),
Tram.Getter());
}
}
```

- Atrybuty: własne lub wykorzystanie istniejących

Atrybuty powołane są w oddzielnej klasie Parameters.cs i dla każdego środka transportu są one wpisywane przy użyciu konstruktora przed klasą:

```
//Powołanie parametrow dla klasy Bus - Punkt w Scorecard: Atrybuty własne
[Parameters(0.5, 0.8, 0.2, 1.2, 1.5, 1.7, 1.3, 6, 00)]
```

```
//Utworzenie parametrow dla klas - Punkt w Scorecard: Atrybuty własne
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]

public class Parameters : Attribute
{
    public double ChanceOfDelay { get; set; }
    public double ChanceOfSmallDelay { get; set; }
    public double ChanceOfBigDelay { get; set; }
    public double SmallDelay { get; set; }
    public double BigDelay { get; set; }
    public double DelayDuringPeakHours { get; set; }
    public double DelayDuringNormalHours { get; set; }

    public int StartHour_H { get; set; }
    public int StartHour_Min { get; set; }

    public int StartOfPeakHours_H { get; set; } = 16;
    public int StartOfPeakHours_Min { get; set; } = 00;
    public int EndOfPeakHours_H { get; set; } = 18;
    public int EndOfPeakHours_Min { get; set; } = 00;

    //Zdefiniowanie wzoru do czytania pliku - Punkt w Scorecard:
Wykorzystanie wyrazen regularnych
    public string pattern { get; set; } = @"[0-9]{1,2}";

    public Parameters(double ChanceOfDelay, double ChanceOfSmallDelay,
double ChanceOfBigDelay, double SmallDelay,
double BigDelay, double DelayDuringPeakHours, double
DelayDuringNormalHours, int StartHour_H, int StartHour_Min)
    {
        this.ChanceOfDelay = ChanceOfDelay;
        this.ChanceOfSmallDelay = ChanceOfSmallDelay;
        this.ChanceOfBigDelay = ChanceOfBigDelay;
    }
}
```

```

        this.SmallDelay = SmallDelay;
        this.BigDelay = BigDelay;
        this.DelayDuringPeakHours = DelayDuringPeakHours;
        this.DelayDuringNormalHours = DelayDuringNormalHours;
        this.StartHour_H = StartHour_H;
        this.StartHour_Min = StartHour_Min;
    }
}

```

- Delegaty: metody anonimowe, wyrażenia lambda

Wyrażenia lambda w tym programie zostały głównie wykorzystane razem z LINQ przy wybieraniu interesujących nas fragmentów pliku np.:

```

IEnumerable<string> Stops = filelist.GetFilesList(path).Where(S => S[0] != 'A');

```

Dodatkowo wykorzystaliśmy metodę anonimową w każdej klasie środka transportu do sprawdzania czy aktualnie jest godzina szczytu przy określeniu opóźnienia:

```

//Zdefiniowanie funkcji anonimowej - Punkt w Scorecard: Metody anonimowe
Func<double, bool> check = (x) => x >= ((parameters.StartOfPeakHours_H * 60) + parameters.StartOfPeakHours_Min)
                                     && x <= ((parameters.EndOfPeakHours_H * 60) + parameters.EndOfPeakHours_Min);

```

```

//Wywołanie funkcji anonimowej - Punkt w Scorecard: Metody anonimowe
Delay = parameters.SmallDelay * (check(CurrentTime) ?
parameters.DelayDuringPeakHours : parameters.DelayDuringNormalHours);

```

- Kolekcje danych i Language Integrated Query

LINQ w tym programie zostały głównie wykorzystane przy wybieraniu interesujących nas fragmentów pliku np.:

```

if (UserLineNum.First() == 'A')
{
    End = false;
    CalculationOfDelay();
}

```

- Metody async, synchronizacja await lub Task API

Async pozwala nam na asynchroniczne czytanie rozkładu dla każdego środka transportu niezależnie od siebie. Dodatkowo dzięki await proces oczekuje na zakończenie czytania plików przez poszczególne środki transportu:

```

//Oczekiwanie na zakończenie wszystkich asynchronicznych taskow - Punkt w Scorecard: Metody async, synchronizacja await
await Task.WhenAll(Bus.FileReader(), Metro.FileReader(),
Train.FileReader(), Tram.FileReader());

```

```
//Wykorzystanie typu generycznego i kowariancji - Punkt w Scorecard:
Zaprojektowanie typu generycznego i kowariancja
MyGenericList<MeansOfTransport> meansOfTransport = new
MyGenericList<MeansOfTransport>{Bus,Metro,Train,Tram};
while (!Bus.End && !Metro.End && !Train.End && !Tram.End)
{
    foreach (var meanOfTransport in meansOfTransport)
    {
        meanOfTransport.getIterator += 1;
    }

    //Oczekiwanie na zakończenie wszystkich asynchronicznych taskow - Punkt
w Scorecard: Metody async, synchronizacja await
    await Task.WhenAll(Bus.Getter(), Metro.Getter(), Train.Getter(),
Tram.Getter());
}
}
```

Wcześniej przedstawiona funkcja Getter() dla tego procesu jest implementacją asynchronicznej funkcji.

- Wykorzystanie wyrażeń regularnych

Wyrażenia regularne służą do zwięzłego wyszukiwania interesujących nas fragmentów linii pliku takich jak np. przystanki i czas przyjazdu między nimi. Wzorzec określa fragmenty stringów pobranych z pliku składających się z jednej lub dwóch cyfr:

```
//Zdefiniowanie wzoru do czytania pliku - Punkt w Scorecard: Wykorzystanie
wyrazen regularnych
public string pattern { get; set; } = @"[0-9]{1,2}";
```

```
//Wykozystanie wyrazen regularnych - Punkt w Scorecard: Wykozystanie
wyrazen regularnych
MatchCollection matches = Regex.Matches(stop, parameters.pattern);
Stop = matches[0].ToString();
TimeBetweenStops = matches[1].ToString();
GivenTimetable.Add(Stop, TimeBetweenStops);
```

- Implementacja i prawidłowe wykorzystanie interfejsu IDisposable

Zastosowanie interfejsu IDisposable pozwala na zdefiniowanie metody Dispose(), dzięki której jesteśmy w stanie pozbyć się nieużywanych już plików, służących do komunikacji międzyprocesowej (Informacja o zakończeniu działania procesu) – wykorzystywane w każdym pliku środka transportu:

```
//Zdefiniowanie funkcji Dispose - Punkt w Scorecard: Implementacja
interface IDisposableable
public void Dispose()
{
    string EndData = $"{directory}EndDataA.txt";
    File.Delete(EndData);
}
}
```

- Wykorzystanie synchronization primitives

Synchronization primitives w naszym wypadku Monitor, służy do synchronizacji dwóch tasków służących do czytania plików z rozkładami jazdy dla konkretnych środków transportu. Task\_1 jako pierwszy wchodzi do monitora i czyta nazwę linii, następnie zapisuje ją do zewnętrznej zmiennej i zwalnia monitor. Task\_2 następnie wchodzi do monitora i czyta rozkład jazdy dla danej linii po czym zapisuje go i łączy z wcześniej odczytaną nazwą linii(Task\_1) w słowniku. Na koniec zwalnia monitor i cała sekwencja rozpoczyna się na nowo aż do przeczytania całego pliku.

```
//Asynchroniczne wywołanie funkcji FileReader() - Punkt w Scorecard:
Metody async
await Task.Run(() =>
{
    string FileName = "Bus.txt";
    string path = $"{directory}{FileName}";
    bool turn = false;
    string Key = "";
    string Stop = "";
    string TimeBetweenStops = "";

    //Wykozystanie zaprojektowanego typu generycznego - Punkt w Scorecard:
    Zaprojektowanie typow generycznych
    //Wykozystanie typow wyliczeniowych - Punkt w Scorecard: Typy
    wyliczeniowe
    //Wykozystanie LINQ - Punkt w Scorecard: Kolekcje danych i Language
    Integrated Query
    //Wykorzystanie wyrażenia lambda - Punkt w Scorecard: Wyrażenia Lambda
    var filelist = new MyGenericList<string>();
    IEnumerable<string> Lines = filelist.GetFilesList(path).Where(L => L[0]
== 'A');
    IEnumerable<string> Stops = filelist.GetFilesList(path).Where(S => S[0]
!= 'A');

    Dictionary<string, string> GivenTimetable = new Dictionary<string,
string>();

    //Wykorzystanie monitora do synchronizacji dzialania obu taskow - Punkt
w Scorecard: Wykorzystanie synchronization primitives
    Task task_1 = Task.Run(() =>
    {
        foreach (var line in Lines)
        {
            lock (_Monitor)
            {
                while (turn)
                {
                    Monitor.Wait(_Monitor);
                }

                Key = line;
                turn = true;
                Monitor.Pulse(_Monitor);
            }
        }
    });

    Task task_2 = Task.Run(() =>
    {
```



```

        foreach (var stop in Stops)
        {
            lock (_Monitor)
            {
                while (!turn)
                {
                    Monitor.Wait(_Monitor);
                    GivenTimetable.Clear();
                }

                if (stop.StartsWith("="))
                {
                    Timetable.Add(Key, GivenTimetable.ToDictionary(entry =>
entry.Key, entry => entry.Value));
                    turn = false;
                    Monitor.Pulse(_Monitor);
                }
                else
                {
                    //Wykozystanie wyrazen regularnych - Punkt w Scorecard:
                    Wykozystanie wyrazen regularnych
                    MatchCollection matches = Regex.Matches(stop,
parameters.pattern);
                    Stop = matches[0].ToString();
                    TimeBetweenStops = matches[1].ToString();
                    GivenTimetable.Add(Stop, TimeBetweenStops);
                }
            }
        }
    });
    Task.WaitAll(task_1, task_2);
});

```