

## Biuro Podróży

0. Stwórz projekt – aplikacja konsolowa lub WPF (przemyśl wybór, bo zmiana może być czasochłonna).

1. Stwórz abstrakcyjną klasę `SrodekLokomocji`.

a) dodaj w niej pola `iloscMiejsc` (int) oraz `cenaBiletu`(double) z mod. `protected`

b) dodaj konstruktor domyślny (pusty)

c) dodaj metodę wirtualną `ObliczCene()` typu void, bez parametru, która za pole `cenaBiletu` podstawia wartość liczbę całkowitą typu int odpowiadającą aktualnemu dniu miesiąca (np. dziś ma mieć wartość 11, jutro 12),

d) dodaj zwykłą metodę zwracającą cenę biletu.

2. Stwórz klasę `Autobus` dziedziczącą z klasy `SrodekLokomocji`. W klasie potomnej wykonaj następujące czynności:

a) dodaj konstruktor parametryczny z parametrem typu int, parametr ma być podstawiony w pole `iloscMiejsc`, w konstruktorze należy wywołać metodę `ObliczCene()` – w wersji z klasy bazowej

b) przesłoń metodę `ToString()` tak, aby zwracała informacje o Autobusie np.  
Autobus: ilość miejsc: 50, cena biletu: 50.

3. Stwórz klasę `Pociąg` dziedziczącą z klasy `SrodekLokomocji`. W klasie potomnej wykonaj następujące czynności:

a) dodaj prywatne pole `dlugoscTrasy` typu int,

b) przesłoń metodę `ObliczCene()` następująco: jeśli długość trasy jest większa niż 100 za cenę biletu należy podstawić wartość  $dlugoscTrasy * 1,43$ ; w przeciwnym wypadku należy podstawić 62.

c) dodaj konstruktor parametryczny z dwoma parametrami (oba typu int), parametry należy podstawić odpowiednio do pól `iloscMiejsc` i `dlugoscTrasy`, w konstruktorze należy również wywołać metodę `ObliczCene()` – w wersji przesłoniętej

d) przesłoń metodę `ToString()` tak, aby zwracała informacje o pociągu:

Pociąg: ilość miejsc: 50, długość trasy: 200, cena biletu: 286.

4. Stwórz dwa interfejsy:

a) `IZarządzaj` – w nim dodaj deklarację następujących metod (wszystkie typu `void`):

`DodajAutobus(int iloscMiejsc)`

`DodajPociag(int iloscMiejsc, int dlugoscTrasy)`

`UsunOstatni()`

`Wyczysc()`

b) `IData` – w nim deklarację metod:

`UstawDate(DateTime data)` – typ zwracany `void`

`SprawdzDate()` – typ zwracany `bool`

5. Stwórz klasę `Podroz`. W tej klasie wykonaj następujące czynności:

a) dodaj prywatne pole `dataPodrozy` typu `DateTime`

b) dodaj prywatne pole `planPodrozy` typu `List<SrodekLokomocji>` (lista przechowująca środki lokomocji), zadбай o inicjację pola

c) dodaj prywatne pole `koszt` typu `double` i nadaj mu wartość początkową 0 (zero)

d) dodaj implementację metod z interfejsów z punktu 5 i podepnij oba interfejsy do klasy `Podroz`, zasady implementacji:

- `DodajAutobus` – dodaje obiekt typu `Autobus` do pola `planPodrozy`, dodatkowo powiększa koszt o cenę biletu

- `DodajPociag` – dodaje obiekt typu `Pociag` do pola `planPodrozy`, dodatkowo powiększa koszt o cenę biletu

- `UsunOstatni` – usuwa ostatni element na liście `planPodrozy`

- `Wyczysc` – usuwa wszystkie elementy z listy `planPodrozy`

- `UstawDate` – ustawia pobrany parametr jako pole `dataPodrozy`

- `SprawdzDate` – zwraca `true` kiedy wartość pola `dataPodrozy` jest większa niż aktualna data pobrana z systemu (można wykorzystać normalny porządek `<` w klasie `DateTime`); w przeciwnym wypadku zwraca `false`

e) przesłoń metodę `ToString()` tak, aby zwracała w kolejnych wierszach informacje o elementach na liście `planPodrozy`

6. Stwórz aplikację WPF lub konsolową do testowania powyższych metod.

## Logika aplikacji:

- należy wykorzystać metody i klasy stworzone w punktach 1-5
- dodawanie do planu podróży Autobusu i Pociągu musi być swobodne/elastyczne tzn. można dodać same autobusy, same pociągi lub na przemian w dowolnej kolejności
- przed dodaniem na listę (planPodrozy) należy upewnić się, że parametry liczbowe są liczbami dodatnimi wskazanego typu
- przy ustawieniu daty podróży należy użytkownikowi przekazać informację na temat wpisywanego formatu daty (np. dd.mm.yyyy lub yyyy/mm/dd, itp.) i aplikacja ma obsłużyć możliwe wyjątki przy zmianie typu string na DateTime
- opcjonalnie użytkownik może wpisać sam dzień (i wtedy domyślnie za czas podstawy się północ) lub użytkownik może podać i datę i godzinę podróży
- sprawdzenie daty podróży ma wyświetlić komunikat np. MessageBoxa lub info na konsoli

## Punktacja

Polecenia 1-4 po 1 pkt każde.

Polecenie 5 – 2pkt

Polecenie 6

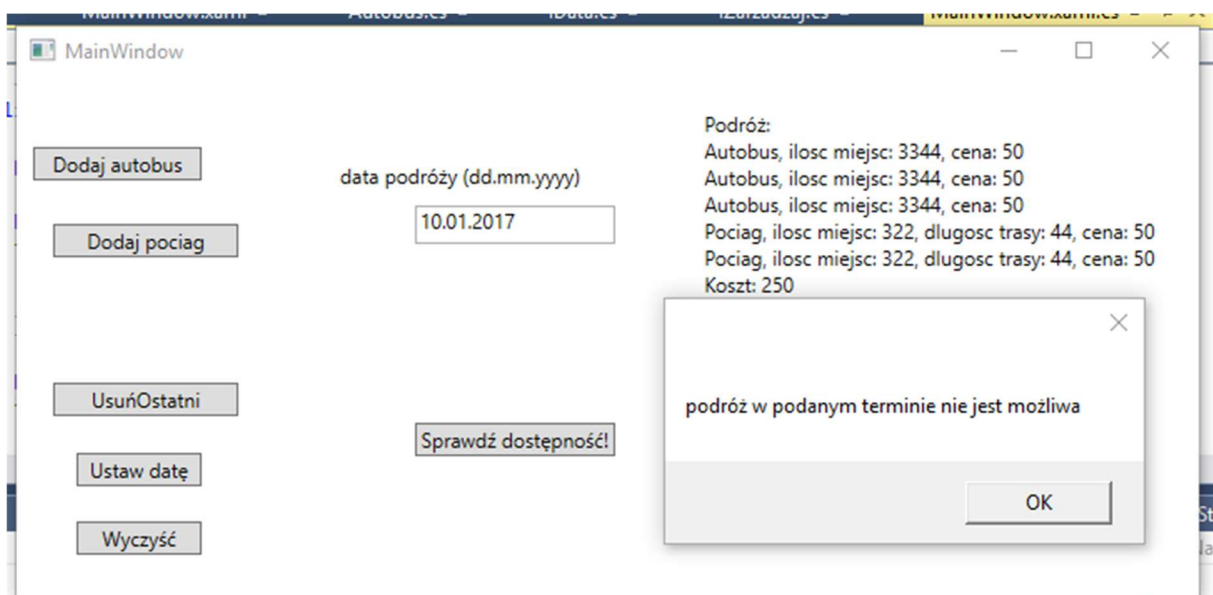
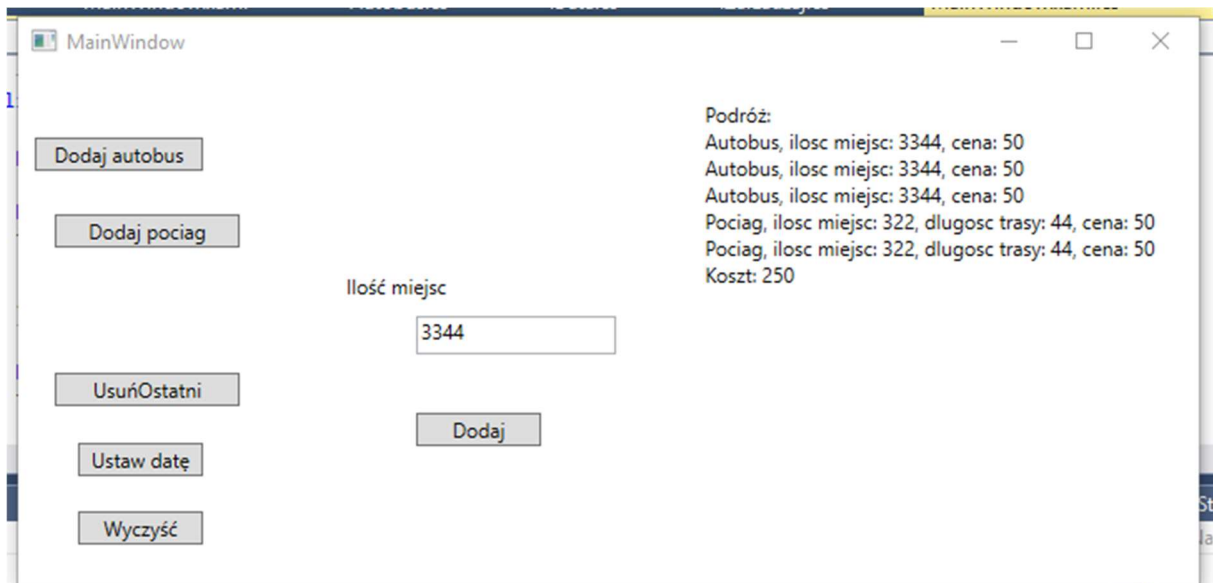
+1 pkt za poprawne działanie kontrolek (tylko dla WPF)

+1 pkt za poprawny algorytm aplikacji przechodzenia po menu (tylko konsola)

+ 1 punkt za obsługę możliwych wyjątków przy zmianie typów i usuwaniu elementów z pustej listy

+ 2 za logikę aplikacji

## Przykładowy screeny wpf



## Przykładowy screen konsola:

```
esZaplanuj swoją podróż!  
[A] - dodaj autobus  
m.[P] - dodaj pociąg  
nsc[U] - usuń ostatnią pozycje z planu podrozy  
1 [Z] - pokaz plan podróży  
2 [D] - sprawdź date podróży  
3
```

## Diagramy UML

