

## 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 `iloscMiejsc * 3`,

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: 24, cena biletu: 72.

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

a) dodaj prywatne pole `odleglosc` typu `int`,

b) przesłoń metodę `ObliczCene()` następująco: jeśli `odleglosc` jest większa niż 200 za cenę biletu należy podstawić wartość `odleglosc * 2,56`; w przeciwnym wypadku należy za cenę biletu podstawić wartość 24.

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

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

Samolot: ilość miejsc: 50, odległość: 200, cena biletu: 24.

4. Stwórz dwa interfejsy:

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

DodajAutobus(int iloscMiejsc)

DodajSamolot(int iloscMiejsc, int odleglosc)

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 200 (dwieście)

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

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

- DodajSamolot – dodaje obiekt typu Pociąg do listy planPodrozy, dodatkowo powiększa koszt o cenę biletu

- UsunOstatni – usuwa ostatni element na liście planPodrozy

- Wyczysc – usuwa wszystkie elementy ze liście 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 Samolotu musi być swobodne/elastyczne tzn. można dodać same autobusy, same samoloty 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

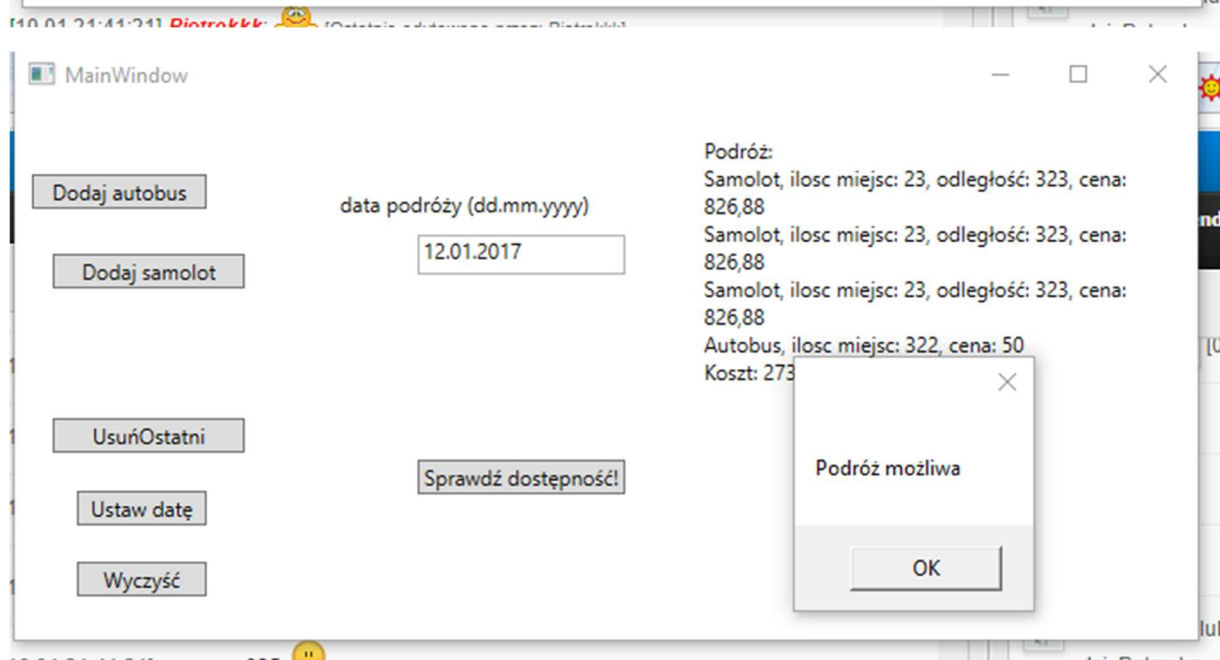
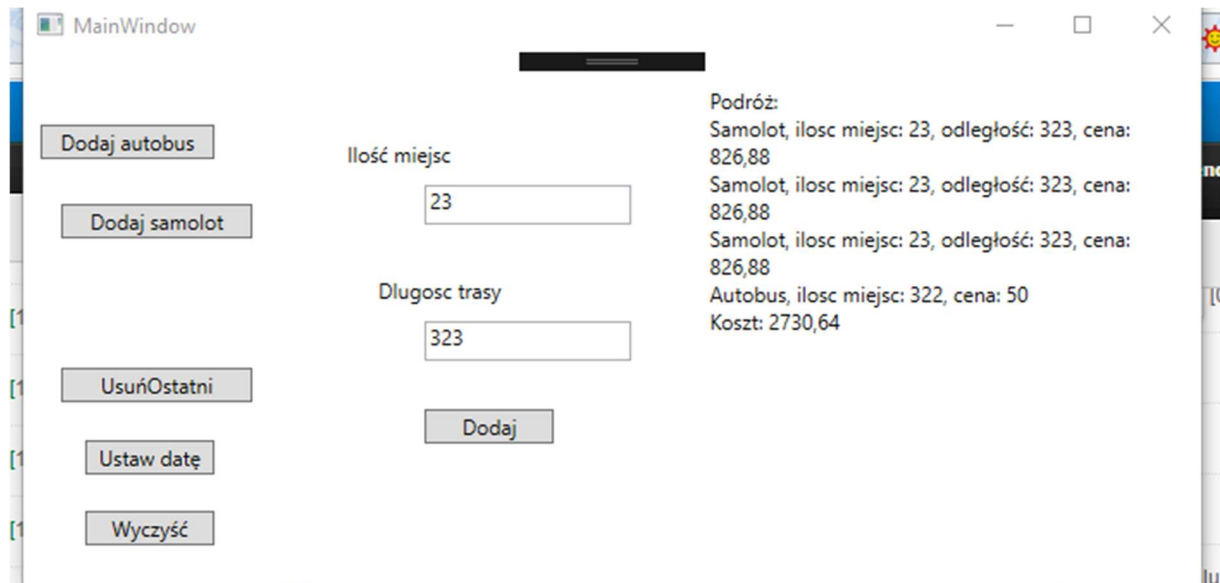
+1 pkt za poprawne działanie kontrolek (tylko dla WPF)

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

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

+ 2 za logikę aplikacji

## Przykładowy screeny wpf



## Przykładowy screen konsola:

```
file:///c:/users/piotrek/documents/visual studio 2013/Projects/C
ca Zaplanuj swoją podróż!
us [A] - dodaj autobus
us [P] - dodaj samolot
na [U] - usuń ostatnią pozycję z planu podróży
[D] - sprawdź datę podróży
```

## Diagramy UML

