Opracowanie: dr inż. Łukasz Maliński, Politechnika Śląska, 2022

LABORATORIUM 2.4
DZIEDZICZENIE I POLIMORFIZM

Zadanie podstawowe [2,0 pkt.]

■ Dla klas **SamochodOsobowy** i **SamochodCiezarowy** z pliku „klasy.h” wydziel klasę bazową **Pojazd**, która zawierać będzie wszystkie wspólne informacje dla obu klas.

Wskazówki: Napisz nową klasę. Klasa Pojazd nie powinna po niczym dziedziczyć, za to klasy już napisane powinny dziedziczyć po klasie Pojazd. Z klas już napisanych usuń te składniki, które zostały wydzielone do klasy Pojazd. Bardzo ważne jest aby nie zmieniać nazw tych składników (szczególnie składników publicznych) aby zachować zgodność interfejsu tych klas. Trzymaj się zasady, że jeśli nie podano jak należy dziedziczyć, to zawsze domyślnie dziedziczymy publicznie.

■ Dla całej hierarchii klas: **Pojazd** i klas pochodnych, utwórz funkcjonalność polimorficzną „opłata rejestracyjna”, której zadaniem będzie wyliczenie kosztu rejestracji danego typu pojazdu według następujących wytycznych:

- **Samochód Osobowy**: *koszt = 500 + 50 * liczba pasażerów + 5 * nadmiarowa prędkość* (nadmiarowa prędkość to każdy km/h prędkości maksymalnej powyżej 140).
- **Samochód Ciężarowy**: koszt = 1000 + pojemność + taryfa typu towaru (sypki – 200, ciekły – 400, paczkowany – 100).

Nie ma zasad wyliczania kosztu dla samego **Pojazdu**, więc przekształć tę klasę w klasę abstrakcyjną.

Wskazówki: Pamiętaj, że aby metoda działała polimorficznie to w klasie bazowej musi mieć odpowiedni specyfikator. Przy redefinicjach metod w klasach pochodnych koniecznie stosuj specyfikator override aby upewnić się, że metody te na pewno przesłaniają oryginalną metodę z klasy bazowej. Przypomnij sobie co należy zrobić aby klasa była klasą abstrakcyjną – tu jest to bardzo intuicyjne gdyż nie definiujemy sposobu wyznaczania kosztu rejestracji dla klasy Pojazd.

■ W programie głównym utwórz tablicę 4 **Pojazdów** i przypisz do niej po 2 **Samochody Osobowe** i **Samochody Ciężarowe**. Następnie napisz krótki kod wyświetlający koszty rejestracji wszystkich pojazdów zebranych w tej tablicy (każdy koszt osobno).

Wskazówki: Zauważ, że nie da się utworzyć tablicy instancji klasy Pojazd, gdyż ta jest klasą abstrakcyjną. Nie jest to problem, gdyż i tak zależy nam na pokazaniu tu działania poimorfizmu. Przypomnij sobie w jaki sposób musimy odwołać się do obiektu aby zadziałał polimorfizm – to odpowie na pytanie jak należy utworzyć tablicę. Wyliczanie kosztów dla wszystkich pojazdów to wywołanie metody polimorficznej dla każdego z tych pojazdów.

Ocenianie:

Wydzielenie klasy bazowej:	0,6 pkt.
Funkcjonalność polimorficzna:	0,6 pkt.
Działanie polimorfizmu w main:	0,6 pkt.
Ogólna jakość kodu:	0,2 pkt.

Zadanie ambitne [2,0 pkt.]:

■ Dla klas **Archer** i **Footman** wydziel klasę bazową **Fighter**. Uczyń klasę **Fighter**, klasą abstrakcyjną. Nowa klasa musi mieć konstruktor argumentowy.

Wskazówki: Wydziel tylko wspólne dane i funkcjonalności. Zastanów się co zrobić aby klasa Fighter była abstrakcyjna mimo, że nie mamy żadnych funkcjonalności polimorficznych. Koniecznie zdefiniuj konstruktor argumentowy dla klasy Fighter i użyj go w konstruktorach klas pochodnych.

■ Dodaj do hierarchii klas klasę **Ranger**, która będzie klasą pochodną zarówno do klasy **Archer** jak i **Footman**. W nowej klasie nie ma potrzeby dodawać żadnych nowych funkcjonalności ale trzeba zdefiniować argumentowy sposób tworzenia instancji.

Wskazówki: Zauważ, że wszystkie 4 klasy tworzą hierarchię diamentu – przypomnij sobie co trzeba zrobić, aby takie dziedziczenie było możliwe, bez podwójnego dziedziczenia klasy korzenia (Fighter). Ważnym aspektem jest właściwe zdefiniowanie listy inicjalizacyjnej konstruktora argumentowego nowej klasy.

■ W main napisz kod, w którym utworzysz tablicę kompania, składającą się z przynajmniej 5 różnych instancji powyższych klas. Każda klasa oprócz klasy **Fighter** powinna w tej tablicy być reprezentowana przynajmniej przez jedną instancję.

Wskazówki: Pamiętaj, że klasa bazowa hierarchii jest abstrakcyjna – nie da się tworzyć jej instancji - mimo to musimy użyć jej typu do tworzenia tablicy, aby dało się przypisać do niej instancje klas pochodnych.

■ Napisz funkcję, która policzy i zwróci przez return ile w kompanii jest instancji poszczególnych klas: **Archer**, **Footman**, **Ranger**.

Wskazówki: Przypomnij sobie jak można użyć rzutowania do identyfikacji typu, oraz jak zwrócić przez return więcej niż jedną wartość.

Ocenianie:

Wydzielenie klasy Fighter:	0,4 pkt.
Dodanie klasy Ranger:	0,6 pkt.
Utworzenie kompanii:	0,4 pkt.
Funkcja zliczająca:	0,4 pkt.
Ogólna jakość kodu:	0,2 pkt.