Języki programowania, Zadanie 9

Dziedziczenie
Dziedziczenie jest to technika pozwalająca na definiowanie nowej klasy, przy wykorzystaniu klasy już wcześniej istniejącej. Często się zdarza, że mamy kilka klas podobnych do siebie. Możemy wtedy stworzyć klasę podstawową, zawierającą część wspólną wszystkich klas, a następnie tworzyć klasy pochodne, zgodnie ze zdaniem „Chcę mieć taką samą klasę jak moja klasa podstawowa, z małymi różnicami (zwykle: dodatkami). Różnice te podaję poniżej...".
Dziedziczenie = Tworzenie klas pochodnych

Treść zadania
Tworzymy komis sprzedający używane pojazdy. Na początek przyjmijmy, że sprzedajemy tylko dwa typy pojazdów: samochody (o składnikach kolor, wiek, cena i ilość) oraz motocykle (o składnikach kolor, wiek, cena, pojemność oraz moc). Zamiast tworzyć dwie całkowicie oddzielne klasy zauważamy, że dla obu sprzedawanych przez nas rodzajów pojazdów część składników się pokrywa (mianowicie: kolor, wiek oraz cena). W dodatku nasza firma zastanawia się nad rozpoczęciem sprzedaży ciężarówek, dla których owe trzy składniki również by się pokrywały. By zaoszczędzić na pisaniu tego samego kodu dwa razy (oraz oszczędzić pracy w przyszłości) postanawiamy napisać klasę nadrzędną `Pojazd`, z której klasy `Samochod` oraz `Motocykl` będą dziedziczyć.

Należy napisać klasę **Pojazd** o następujących polach składowych
- `std::string kolor;`
- `int wiek;`
- `double cena;`

z konstruktorem domyślnym oraz konstruktorem z 3 parametrami, oraz metodach:
- `SetCena(double c)`
- `double GetCena()`
- `void Wypisz()`

Oraz klasy:
- **Samochod** dziedziczącą z `Pojazd`, z dodatkowym polem `int iloscDrzwi`. Z konstruktorami: domyślnym (zerującym wszystkie pola) oraz konstruktorem z czterema parametrami. Ponadto klasa powinna mieć metody: „`Wypisz`" (bezargumentową, wypisującą dane samochodu na ekran).
- **Motocykl** dziedziczącą z `Pojazd`, z dodatkowymi polami `int pojemnosc` (w litrach), oraz `int moc` (w KM). Z konstruktorami: domyślnym (zerującym wszystkie pola), z trzema parametrami (podstawowymi dla klasy `Pojazd`, czyli kolorem, wiekiem i ceną, ustawianymi przez konstruktor klasy nadrzędnej, reszta jest zerowana) oraz z pięcioma parametrami. Wszystkie konstruktory powinny ustawiać odpowiednie pola przy użyciu listy inicjalizacyjnej. Ponadto należy przeciążyć metodę Wypisz, używając metody o tej samej nazwie z klasy bazowej.
- **Kabriolet** dziedziczącą z Samochód, z dodatkowym polem enum Dach (miekki_manualny=0, miekki_automatyczny=1, twardy_manualny=2, twardy_automatyczny=3). Należy dodać metodę Wypisz która słownie wypisze na ekran własności samochodu oraz typ dachu. Również powinna wykorzystywać kod klas bazowych.

W głównej funkcji programu należy stworzyć po dwa obiekty klasy `Samochod`, oraz trzy obiekty klasy `Motocykl`, przy użyciu różnych konstruktorów.
Następnie należy stworzyć po trzy wskaźniki na takie obiekty, również przy użyciu różnych konstruktorów.

Należy postępować według punktów:
1. Należy stworzyć osobny katalog, a w nim 9 pustych plików tekstowych: **program.cpp**, **pojazd.cpp**, **pojazd.h**, **samochod.cpp**, **samochod.h**, **motocykl.cpp**, **motocykl.h, kabriolet.cpp, kabriolet.h**. Następnie należy stworzyć Makefile. W pliku program.cpp należy dodać funkcję `int main()` zwracającą 0.
   Należy skompilować tak przygotowany program używając w linii poleceń komendy: `make`.

2. Następnie tworzymy klasę `Pojazd`.

3. Tworzymy klasę Samochód. Po kolei:
   - Tworzymy klasę Samochod, będącą klasą pochodną klasy Pojazd
     ```
     class nazwa_klasy_pochodnej : public nazwa_klasy_podstawowej {};
     ```

- Zapisujemy, kompilujemy. Poprawiamy błędy. Wskazówki:
  a) W przypadku błędu typu:
  ```
  pojazd.h:5:7: error: redefinition of ‘class pojazd’
  pojazd.h:5:12: error: previous definition of ‘class pojazd’
  ```
  ten błąd występuje, wtedy gdy dwa razy próbujemy dodać tę samą klasę. Kompilator się skarży, że próbujemy drugi raz zdefiniować to samo. Prawidłową metodą radzenia sobie z tym jest owijanie definicji klas w plikach nagłówkowych w strukturę „ifndef”:
  ```
  #ifndef _NAZWA_TOKENU
  #define _NAZWA_ TOKENU
     class klasa{...}; - definicja naszej klasy
  #endif
  ```
  b) W przypadku błędu typu:
  ```
  ‘int pojazd::wiek’ is private
  ```
  Domyślnie wszystkie zmienne **`private`** są niedostępne poza daną klasą, czyli RÓWNIEŻ dla klas pochodnych. W naszym wypadku chcielibyśmy, by te zmienne były dla nas dostępne, ale z drugiej strony - dostępne *tylko* w naszych klasach pochodnych – nie publicznie dostępne na zewnątrz. Takie zmienne powinny zostać zadeklarowane jako **`protected`** w klasie pojazd. Tak więc tutaj wyjaśnia się, do czego służy kwalifikator dostępu `protected` – pola, które w klasie nadrzędnej opatrzone są takim kwalifikatorem, są dostępne w klasach pochodnych ale nie są dostępne poza klasami.

4. Tworzymy klasę `Motocykl`. Postępujemy dokładnie tak samo jak z klasą `Samochod`. Polecane jest skopiowanie klasy `Samochod` oraz odpowiednie pozmienianie nazw i typów.

5. Należy ponadto zapoznać się z treścią pliku: http://www.if.pw.edu.pl/~majanik/data/JP/2012/makefile.pdf.
   Napisany **Makefile**, powinien umożliwiać kompilację napisanego programu, definiować zmienną CXX określającą kompilator (g++) oraz CXXFLAGS definiującą flagę (-Wall), a tworzone klasy powinny być wstępnie kompilowane do plików *.o.