Zadanie 9, Języki Programowania, 4.12.2011

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 sklep sprzedający używany sprzęt elektroniczny. Na początek przyjmijmy, że sprzedajemy tylko dwa typy sprzętu: telewizory (o składnikach kolor, wiek, cena i przekątna) oraz mikrofalówki (o składnikach kolor, wiek, cena, pojemność oraz moc). Zamiast tworzyć dwóch całkowicie oddzielnych klas zauważamy, że dla obu sprzedawanych przez nas rodzajów sprzętu część składników się pokrywa (mianowicie: kolor, wiek oraz cena). W dodatku nasza firma zastanawia się nad rozpoczęciem sprzedaży odtwarzaczy mp3, dla których owe trzy składniki również by się pokrywały. By zaoszczędzić na pisaniu tego samego dwa razy (oraz oszczędzić pracy w przyszłości) postanawiamy napisać klasę nadrzędną Towar z której klasy Telewizor oraz Mikrofalówka będą dziedziczyć.

Należy napisać klasę **Towar** o następujących polach składowych

- `std::string kolor;`
- `int wiek;`
- `double cena;`

Oraz konstruktorem z 3 parametrami, oraz metodach:

- `SetKolor(std::string k)`
- `SetCena(double c)`
- `SetWiek(int w)`
- `std::string GetKolor()`
- `double GetCena()`
- `int GetWiek()`
- `void Wypisz()`

Oraz klasy:

- **Telewizor** dziedziczącą z Towaru, z dodatkowym polem `double przekatna`. Z konstruktorami: domyślnym (zerującym wszystkie pola) oraz konstruktorem z czterema parametrami. Ponadto klasa telewizor powinna mieć metody: „`Wypisz`" (bezargumentową, wypisującą dane telewizora na ekran) oraz `double GetPrzekatna()` i `void SetPrzekatna(double p)`.
- **Mikrofalówka** dziedziczącą z Towaru, z dodatkowymi polami „int pojemność" (w litrach), oraz „int moc" (w kW). Z konstruktorami: domyślnym (zerującym wszystkie pola), z trzema parametrami (podstawowymi dla klasy Towar, czyli kolorem, wiekiem i ceną, reszta jest zerowana) oraz z pięcioma parametrami.

Trzeba zauważyć, że wszystkie klasy i metody są bardzo podobne – posiadają jedynie funkcje typu „get" i „set" oraz, w przypadku klas `Telewizor` i `Mikrofalówka`, konstruktory (domyślny i z parametrami). Zadaniem funkcji typu „set" jest ustawienie odpowiednich pól składowych, tzn.:
załóżmy, że mamy w naszej klasie pole `int fIlosc`. Wtedy metody „set" i „get" dla takiego pola:

```
void SetIlosc(int ilosc) //metoda typu „set" / „setter"
{
  fIlosc = ilosc;
}


int GetIlosc() //metoda typu „get" / „getter"
{
  return fIlosc;
}
```

W głównej funkcji programu należy stworzyć po trzy obiekty klasy Telewizor, oraz trzy obiekty klasy Mikrofalówka, 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 7 pustych plików tekstowych: **program.cpp**, **towar.cpp**, **towar.h**, **telewizor.cpp**, **telewizor.h**, **mikrofalowka.cpp**, **mikrofalowka.h**. Następnie należy stworzyć Makefile (kolejny plik tekstowy, o nazwie „**Makefile**") z treścią:
   ```
   all:
   	g++ -Wall towar.cpp telewizor.cpp mikrofalowka.cpp program.cpp -o program
   ```
   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 towar. Po kolei:
   - W pliku nagłówkowym (.h) deklarujemy odpowiednie składniki i metody.
   - Wypełniamy plik (.cpp). Dobrą praktyką jest skopiować deklaracje funkcji z pliku h. Pamiętamy o dodaniu zakresu (towar::).
   - Tworzymy obiekt klasy towar w głównej funkcji programu.
     ```
     nazwa_klasy nazwa_obiektu;
     ```
   - Używamy na nim napisanych przez nas metod. Metody wywołujemy:
     ```
     nazwa_obiektu.nazwa_metody(argumenty);
     ```
   - Kompilujemy (`make`!) i uruchamiamy.
3. Tworzymy klasę telewizor. Po kolei:
   - Tworzymy klasę telewizor, będącą klasą pochodną klasy towar
     ```
     class nazwa_klasy_pochodnej : public nazwa_klasy_podstawowej {};
     ```
   - Deklarujemy dodatkowe składniki, konstruktory oraz metody w pliku h.
   - Deklarujemy odpowiednie konstruktory i metody w pliku cpp. Na razie wypełniamy tylko metodę `Wypisz`, konstruktory pozostawiamy puste.
   - W głównym programie tworzymy obiekt klasy telewizor, następnie wypisujemy go na ekran, używając metody `Wypisz()`.
   - Zapisujemy, kompilujemy. Poprawiamy błędy. Wskazówki:
     a) ZAWSZE poprawiamy najpierw pierwszy błąd na liście.
     b) W przypadku błędu typu:
     ```
     towar.h:5:7: error: redefinition of 'class towar'
     towar.h:5:12: error: previous definition of 'class towar'
     ```
     to jest błąd, o istnieniu którego opowiadałam na poprzednich zajęciach – jeśli 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
     ```
     c) W przypadku błędu typu:
     ```
     'int towar::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 towar. 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.
   - Deklarujemy jeszcze wskaźnik, oraz na nim wywołujemy metodę Wypisz.
   - Wypełniamy konstruktory. Po jednym naraz: wypełniamy jeden, deklarujemy obiekt używając danego konstruktora, uruchamiamy program. Dopiero wtedy wypełniamy następny. W ogólności: trzeba wszystkim składnikom przypisać odpowiednie wartości (odpowiednie = podane jako argumenty, albo 0, lub dla string: " ")
   - Wołamy prowadzącego, żeby pochwalić się, iż już działa.
4. Zamieniamy konstruktor bezparametrowy dla klasy telewizor w ten sposób, by użyta została lista inicjalizacyjna konstruktora. Uwaga! W tym celu należy dopisać odpowiedni konstruktor bezparametrowy dla klasy towar, a następnie wywołać go w liście inicjalizacyjnej konstruktora `Telewizor()`.
5. Tworzymy klasę mikrofalówka. Postępujemy dokładnie tak samo jak z klasą telewizor. Polecane jest skopiowanie klasy telewizor oraz odpowiednie pozmienianie nazw i typów.