

Języki Programowania, Zadanie 7 – realizowane w parach

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 komputery. Na początek przyjmijmy, że sprzedajemy dwa typy komputerów: stacjonarne (o składnikach nazwa, wiek, cena i karta graficzna) oraz laptopy (o składnikach nazwa, wiek, cena, masa oraz czas pracy baterii). Zamiast tworzyć dwie całkowicie oddzielne klasy zauważamy, że dla obu sprzedawanych przez nas rodzajów komputerów część składników się pokrywa (mianowicie: nazwa, wiek oraz cena). W dodatku nasza firma zastanawia się nad rozpoczęciem sprzedaży tabletów, 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 `Komputer` z której klasy `Desktop` oraz `Laptop` będą dziedziczyć.

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

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

Oraz konstruktorem domyślnym i głównym z 3 parametrami, oraz metodach:

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

Oraz klasy:

- **Desktop** dziedziczącą z `Komputer`, z dodatkowym polem `string kartaGraficzna`. Z konstruktorami: domyślnym (ustawia pola na domyślne wartości) oraz konstruktorem z czterema parametrami. Ponadto klasa `Desktop` powinna mieć metody: „Wypisz” (bezargumentową, wypisującą dane komputera na ekran) oraz `string GetKartaGraficzna()` i `void SetKartaGraficzna(string w)`. **Nie używamy listy inicjalizacyjnej konstruktora.**
- **Laptop** dziedziczącą z `Komputer`, z dodatkowymi polami `double masa` (w kilogramach), oraz `int czasBaterii` (w minutach). Z konstruktorami: domyślnym (ustawia wszystkie pola na domyślne wartości), z trzema parametrami (podstawowymi dla klasy `Komputer`, czyli nazwą, wiekiem i ceną, reszta jest domyślna) oraz z pięcioma parametrami. Ponadto należy przeciążyć metodę `Wypisz`, używając metody o tej samej nazwie z klasy bazowej. **Należy użyć listy inicjalizacyjnej konstruktora do ustawienia pól składowych w konstruktorach.**

Trzeba zauważyć, że wszystkie klasy i metody są bardzo podobne – posiadają jedynie funkcje typu „get” i „set” oraz, w przypadku klas `Desktop` i `Laptop`, konstruktory (domyślny i z parametrami).

W głównej funkcji programu należy stworzyć

- obiekt klasy `Komputer` przy użyciu konstruktora z 3 parametrami.
- po trzy obiekty klasy `Desktop` przy użyciu różnych konstruktorów.
- trzy obiekty klasy `Laptop` 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** (lub **main.cpp**), **komputer.cpp**, **komputer.h**, **desktop.cpp**, **desktop.h**, **laptop.cpp**, **laptop.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 `Komputer`.
3. Tworzymy klasę `Desktop`.
 - Tworzymy klasę `Desktop`, będącą klasą pochodną klasy `Komputer`

```
class nazwa_klasy_pochodnej : public nazwa_klasy_podstawowej {};
```
 - Zapisujemy, kompilujemy. Poprawiamy błędy. Wskazówki:
 - a) ZAWSZE poprawiamy najpierw pierwszy błąd na liście.
 - b) W przypadku błędu typu:

```
komputer.h:5:7: error: redefinition of 'class komputer'
komputer.h:5:12: error: previous definition of 'class komputer'
```

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
```
 - c) W przypadku błędu typu:

```
'int komputer::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.
 - 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: `""`)
4. Tworzymy klasę `Laptop`.
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`. Należy tak skonfigurować Geany, by móc kompilować przez cegiełkę przy użyciu stworzonego `makefile'a` (wystarczy raz wybrać z rozwijanej listy przy cegiełce „`Make all`”, by ta opcja wskoczyła jako domyślna).
6. Zamienić klasę `Komputer` w klasę abstrakcyjną (ATD, abstrakcyjny typ danych). Metodą czysto wirtualną powinna zostać metoda `Wypisz`:
 - **Wypisz() = 0**