

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)`.
- **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. Wszystkie konstruktory powinny ustawiać odpowiednie pola przy użyciu listy inicjalizacyjnej. Ponadto należy przeciążyć metodę `Wypisz`, używając metody o tej samej nazwie z klasy bazowej.

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 po trzy obiekty klasy `Desktop`, oraz 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. [Powtórzenie Makefile] Należy zapoznać się z treścią pliku: <http://www.if.pw.edu.pl/~majanik/data/IP/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 cegielkę przy użyciu stworzonego makefile'a (wystarczy raz wybrać z rozwijanej listy przy cegielce „Make all”, by ta opcja wskoczyła jako domyślna).
6. Ponadto, w funkcji main należy utworzyć Komputer (a następnie wypisać go na ekran), którego nazwa, wiek oraz cena podane są jako **parametry wywołania programu**. Uwaga, program powinien wypisywać odpowiednie ostrzeżenie, jeśli liczba podanych parametrów jest nieprawidłowa!

Parametry wywołania programu (zmiany w deklaracji funkcji main):

```
int main(int argc, char **argv){
}
```

Gdzie:

argc - liczba parametrów, z którymi został wywołany program

**argv - tablica parametrów (każdy jako char*):

argv[0] - nazwa programu

argv[1], argv[2]... - kolejne parametry wywołania programu

Np. wywołanie programu program mogło by mieć formę:

```
./program Ala 2 3.5
```

Wtedy:

```
//argc == 4
```

```
//argv[0] == „program”
```

```
//argv[1] == „Ala”
```

```
//argv[2] == „2” || zmiana „2” na 2 (liczbę całkowitą) – funkcja atoi. np int a = atoi(“3”); z biblioteki <stdlib.h>.
```

```
//argv[3] == „3.5” || zmiana „3.5” na 3.5 (liczbę zmiennoprzecinkową) – funkcja double a = atof(“3.5”); z. bibl. <stdlib.h>.
```

Przyporządkowanie tych parametrów wykonuje się samoistnie w momencie wywołania programu, jedynym wkładem programisty jest zadeklarowanie funkcji main w formie int main(int argc, char **argv)